Important Information

SOME TIBCO SOFTWARE EMBEDS OR BUNDLES OTHER TIBCO SOFTWARE. USE OF SUCH EMBEDDED OR BUNDLED TIBCO SOFTWARE IS SOLELY TO ENABLE THE FUNCTIONALITY (OR PROVIDE LIMITED ADD-ON FUNCTIONALITY) OF THE LICENSED TIBCO SOFTWARE. THE EMBEDDED OR BUNLED SOFTWARE IS NOT LICENSED TO BE USED OR ACCESSED BY ANY OTHER TIBCO SOFTWARE OR FOR ANY OTHER PURPOSE.

USE OF TIBCO SOFTWARE AND THIS DOCUMENT IS SUBJECT TO THE TERMS AND CONDITIONS OF A LICENSE AGREEMENT FOUND IN EITHER A SEPARATELY EXECUTED SOFTWARE LICENSE AGREEMENT, OR, IF THERE IS NO SUCH SEPARATE AGREEMENT, THE CLICKWRAP END USER LICENSE AGREEMENT WHICH IS DISPLAYED DURING DOWNLOAD OR INSTALLATION OF THE SOFTWARE (AND WHICH IS DUPLICATED IN THE LICENSE FILE) OR IF THERE IS NO SUCH SOFTWARE LICENSE AGREEMENT OR CLICKWRAP END USER LICENSE AGREEMENT, THE LICENSE(S) LOCATED IN THE "LICENSE" FILE(S) OF THE SOFTWARE. USE OF THIS DOCUMENT IS SUBJECT TO THOSE TERMS AND CONDITIONS, AND YOUR USE HEREOF SHALL CONSTITUTE ACCEPTANCE OF AND AN AGREEMENT TO BE BOUND BY THE SAME.

This document contains confidential information that is subject to U.S. and international copyright laws and treaties. No part of this document may be reproduced in any form without the written authorization of TIBCO Software Inc.

TIBCO, Two-Second Advantage, TIBCO ActiveMatrix, and TIBCO Enterprise Message Service are either registered trademarks or trademarks of TIBCO Software Inc. in the United States and/or other countries.

Enterprise Java Beans (EJB), Java 2 Platform Enterprise Edition (J2EE), and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle Corporation in the U.S. and other countries.

All other product and company names and marks mentioned in this document are the property of their respective owners and are mentioned for identification purposes only.

THIS SOFTWARE MAY BE AVAILABLE ON MULTIPLE OPERATING SYSTEMS. HOWEVER, NOT ALL OPERATING SYSTEM PLATFORMS FOR A SPECIFIC SOFTWARE VERSION ARE RELEASED AT THE SAME TIME. SEE THE README FILE FOR THE AVAILABILITY OF THIS SOFTWARE VERSION ON A SPECIFIC OPERATING SYSTEM PLATFORM.

THIS DOCUMENT IS PROVIDED “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR NON-INFRINGEMENT.

THIS DOCUMENT COULD INCLUDE TECHNICAL INACCURACIES OR TYPOGRAPHICAL ERRORS. CHANGES ARE PERIODICALLY ADDED TO THE INFORMATION HEREIN; THESE CHANGES WILL BE INCORPORATED IN NEW EDITIONS OF THIS DOCUMENT. TIBCO SOFTWARE INC. MAY MAKE IMPROVEMENTS AND/OR CHANGES IN THE PRODUCT(S) AND/OR THE PROGRAM(S) DESCRIBED IN THIS DOCUMENT AT ANY TIME.

THE CONTENTS OF THIS DOCUMENT MAY BE MODIFIED AND/OR QUALIFIED, DIRECTLY OR INDIRECTLY, BY OTHER DOCUMENTATION WHICH ACCOMPANIES THIS SOFTWARE, INCLUDING BUT NOT LIMITED TO ANY RELEASE NOTES AND "READ ME" FILES.

Copyright © 1998-2013 TIBCO Software Inc. ALL RIGHTS RESERVED.

TIBCO Software Inc. Confidential Information
Contents

Figures.................................................................................................................. xi

Tables..................................................................................................................... xiii

Preface.................................................................................................................... xv

Related Documentation........................................................................................ xvi
  TIBCO ActiveMatrix® Service Grid Documentation.......................................... xvi
  Other TIBCO Product Documentation............................................................... xvii

Typographical Conventions................................................................................. xx

Connecting with TIBCO Resources...................................................................... xxi
  How to Join TIBCOmmunity .............................................................................. xxi
  How to Access All TIBCO Documentation........................................................ xxi
  How to Contact TIBCO Support....................................................................... xxi

Chapter 1 Introduction........................................................................................... 1
  Overview of Mediation......................................................................................... 2
  Mediation Flows.................................................................................................. 4
    Message Exchange Patterns.............................................................................. 5
    Mediation Flow Interfaces.............................................................................. 5
    Paths in a Mediation Flow................................................................................ 6
    Mediation Tasks................................................................................................. 8
    Mediation Exchange......................................................................................... 10
  Designing Mediation Flows................................................................................. 12

Chapter 2 Working with Mediation Flows........................................................... 15
  Creating Mediation Flows................................................................................ 16
    Starting the Mediation Flow Wizard.............................................................. 16
    Creating a New, Empty Mediation Flow........................................................ 17
    Creating New Mediation Flows From Existing Web Services........................ 17
    Editing Preferences of the Mediation Flow Editor.......................................... 19
  Mediation Flow Properties................................................................................. 22
  Working with Interfaces...................................................................................... 24
    Adding Interfaces to Mediation Flows............................................................ 24
    Deleting Interfaces.......................................................................................... 26
# Contents

<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chapter 2</td>
<td>Mediation Components</td>
<td>45</td>
</tr>
<tr>
<td>Working with Context Parameters</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>Adding Context Parameters</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Working with path</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>Modifying Paths</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>Working with Message Context Properties</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Adding Context Parameters</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>Working with Context Parameters</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>Mediation Context Parameters</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>Working with Exchange Variables</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>Defining Exchange Variables</td>
<td>37</td>
<td></td>
</tr>
<tr>
<td>Setting the Exchange Variable</td>
<td>38</td>
<td></td>
</tr>
<tr>
<td>Creating Simple Schemas</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td>Working with Tasks</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>Adding a Task to a Path</td>
<td>42</td>
<td></td>
</tr>
<tr>
<td>Deleting a Task From a Path</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>Cutting, Copying, and Pasting Tasks</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td>Mediation Components</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>Working with Mediation Components</td>
<td>46</td>
<td></td>
</tr>
<tr>
<td>Intents and Policies</td>
<td>47</td>
<td></td>
</tr>
<tr>
<td>Chapter 3</td>
<td>Invoking an Operation</td>
<td>49</td>
</tr>
<tr>
<td>Overview of the Invoke Operation Task</td>
<td>50</td>
<td></td>
</tr>
<tr>
<td>Configuring Invoke Operation tasks</td>
<td>51</td>
<td></td>
</tr>
<tr>
<td>Input Tab</td>
<td>51</td>
<td></td>
</tr>
<tr>
<td>Output Tab</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>Fault Handling for Invoke Operations</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>Invoking Operations on Dynamic Interfaces</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>Chapter 4</td>
<td>Logging Mediation Exchange Information</td>
<td>53</td>
</tr>
<tr>
<td>Overview of Log Tasks</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>Configuring a Log Task</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>Using the Standard Log Tab</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>Using the Mapper Log Tab</td>
<td>56</td>
<td></td>
</tr>
<tr>
<td>Configuring Mediation Appenders and Loggers</td>
<td>58</td>
<td></td>
</tr>
<tr>
<td>Chapter 5</td>
<td>Routing Messages in a Mediation Flow</td>
<td>59</td>
</tr>
<tr>
<td>Overview of Route Tasks</td>
<td>60</td>
<td></td>
</tr>
<tr>
<td>Paths and Route Tasks</td>
<td>61</td>
<td></td>
</tr>
<tr>
<td>Defining a Route</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td>Routing Cases</td>
<td>64</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 6 Transforming Data in a Mediation Exchange

Overview of Transform Tasks
Example of Transformation
Contributing Data to the Mediation Exchange
Using External Stylesheets for Data Transformation
   Specifying an External Stylesheet for Data Transformation
Schema Components
Qualifier Characters
Context Panel
Message Panel
Data/Function Tabs
   Creating Custom XPath Functions
   Exporting Custom XPath Functions
   Deploying Custom XPath Functions
   Testing Custom XPath Functions
Toolbar
Right-Click Menu
   Choose Statements
   If Statements
   For Each Statements
   Variables
   Comments
Mappings
   Repairing Incorrect Mappings
   Mapping an Empty Complex Type
Using XPath
   Addressing Schema Components
   Evaluation Context
### Contents

<table>
<thead>
<tr>
<th>Search Predicates</th>
<th>108</th>
</tr>
</thead>
<tbody>
<tr>
<td>Testing Boolean Values</td>
<td>109</td>
</tr>
<tr>
<td>Comments</td>
<td>109</td>
</tr>
<tr>
<td>Transforming XML with Related Tasks</td>
<td>110</td>
</tr>
<tr>
<td><strong>Chapter 7  Querying a Database</strong></td>
<td>111</td>
</tr>
<tr>
<td>Overview of Query Database Tasks</td>
<td>112</td>
</tr>
<tr>
<td>JDBC Resource Templates</td>
<td>113</td>
</tr>
<tr>
<td>Configuring a JDBC Driver</td>
<td>114</td>
</tr>
<tr>
<td>Registering a JDBC Driver</td>
<td>114</td>
</tr>
<tr>
<td>Configuring a Query Database Task</td>
<td>116</td>
</tr>
<tr>
<td>General Tab</td>
<td>116</td>
</tr>
<tr>
<td>Query Tab</td>
<td>116</td>
</tr>
<tr>
<td>Input Tab</td>
<td>118</td>
</tr>
<tr>
<td>Output Tab</td>
<td>118</td>
</tr>
<tr>
<td>Test Tab</td>
<td>119</td>
</tr>
<tr>
<td><strong>Chapter 8  Routing Requests Dynamically at Runtime</strong></td>
<td>121</td>
</tr>
<tr>
<td>Overview of Dynamic Binding</td>
<td>122</td>
</tr>
<tr>
<td>Service Providers for Dynamic Composite References</td>
<td>123</td>
</tr>
<tr>
<td>Configuring Dynamic Binding</td>
<td>124</td>
</tr>
<tr>
<td>Configuring Dynamic Target Interfaces</td>
<td>125</td>
</tr>
<tr>
<td>Marking a Target Interface as Dynamic</td>
<td>125</td>
</tr>
<tr>
<td>Using Pattern Variables</td>
<td>126</td>
</tr>
<tr>
<td>Setting the Dynamic Reference Task</td>
<td>127</td>
</tr>
<tr>
<td>Configuring the General Tab</td>
<td>128</td>
</tr>
<tr>
<td>Specifying the Input</td>
<td>129</td>
</tr>
<tr>
<td>Configuring Dynamic References in Composite</td>
<td>132</td>
</tr>
<tr>
<td>Creating and Deploying Composites Used By Dynamic Binding</td>
<td>133</td>
</tr>
<tr>
<td><strong>Chapter 9  Replying to Messages</strong></td>
<td>135</td>
</tr>
<tr>
<td>Overview of Reply Messages</td>
<td>136</td>
</tr>
<tr>
<td>Working with Reply Messages</td>
<td>138</td>
</tr>
<tr>
<td><strong>Chapter 10  Fault Processing in a Mediation Flow</strong></td>
<td>141</td>
</tr>
<tr>
<td>Overview of Faults</td>
<td>142</td>
</tr>
<tr>
<td>Throwing Faults in Mediation Flows</td>
<td>145</td>
</tr>
<tr>
<td>Working with Fault Paths</td>
<td>147</td>
</tr>
<tr>
<td>Catching Faults From the Target Operation</td>
<td>147</td>
</tr>
<tr>
<td>Catching Faults From the Mediation Flow</td>
<td>148</td>
</tr>
</tbody>
</table>
### Chapter 11 Custom Mediation Tasks

- Overview .................................................................................................................. 152
- Migrating Custom Mediation Tasks ........................................................................... 153
  - Eclipse Plug-ins ........................................................................................................ 155
- Support Files ............................................................................................................... 156
- Creating the Model Plug-in ....................................................................................... 157
- Creating the UI Plug-in ................................................................................................ 162
- Creating the Runtime Plug-in .................................................................................... 165
- Writing Custom Mediation Code ............................................................................... 168
  - Accessing Task Input/Output Schema .................................................................... 168
  - Modifying the Mediation Task Data ...................................................................... 169
  - Defining Model Attributes ..................................................................................... 170
  - Categorizing Custom Mediation Tasks ................................................................ 170
  - Thrown Faults ......................................................................................................... 171
  - Runtime Exceptions ................................................................................................ 171
- Packaging Custom Mediation Tasks .......................................................................... 172
  - Installing Custom Mediation Tasks ....................................................................... 172
  - Deploying Custom Mediation Tasks ...................................................................... 172
  - Testing Custom Mediation Tasks ......................................................................... 173

### Chapter 12 Reference

- Catch Fault .................................................................................................................. 176
  - General Tab ............................................................................................................. 176
- End Mediation ............................................................................................................ 178
  - General Tab ............................................................................................................. 179
- Generate Reply .......................................................................................................... 180
  - General Tab ............................................................................................................. 180
  - Input Tab .................................................................................................................. 180
- Handle Reply ............................................................................................................. 181
  - General Tab ............................................................................................................. 181
- Invoke Operation ......................................................................................................... 182
  - General Tab ............................................................................................................. 182
  - Input Tab .................................................................................................................. 183
  - Output Tab .............................................................................................................. 183
- Log ............................................................................................................................... 184
  - General Tab ............................................................................................................. 184
  - Log Tab ..................................................................................................................... 185
<table>
<thead>
<tr>
<th>Contents</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse XML</td>
<td>188</td>
</tr>
<tr>
<td>General Tab</td>
<td>188</td>
</tr>
<tr>
<td>Input Tab</td>
<td>189</td>
</tr>
<tr>
<td>Output Tab</td>
<td>190</td>
</tr>
<tr>
<td>Query Database</td>
<td>191</td>
</tr>
<tr>
<td>General Tab</td>
<td>191</td>
</tr>
<tr>
<td>Query Tab</td>
<td>191</td>
</tr>
<tr>
<td>Input Tab</td>
<td>194</td>
</tr>
<tr>
<td>Output Tab</td>
<td>194</td>
</tr>
<tr>
<td>Test Tab</td>
<td>194</td>
</tr>
<tr>
<td>Render XML</td>
<td>195</td>
</tr>
<tr>
<td>General Tab</td>
<td>195</td>
</tr>
<tr>
<td>Input Tab</td>
<td>196</td>
</tr>
<tr>
<td>Output Tab</td>
<td>196</td>
</tr>
<tr>
<td>Route Task</td>
<td>197</td>
</tr>
<tr>
<td>General Tab</td>
<td>197</td>
</tr>
<tr>
<td>Decision Tab</td>
<td>198</td>
</tr>
<tr>
<td>Input Tab</td>
<td>200</td>
</tr>
<tr>
<td>Send Fault</td>
<td>201</td>
</tr>
<tr>
<td>General Tab</td>
<td>201</td>
</tr>
<tr>
<td>Set Context</td>
<td>202</td>
</tr>
<tr>
<td>General Tab</td>
<td>202</td>
</tr>
<tr>
<td>Input Tab</td>
<td>203</td>
</tr>
<tr>
<td>Set Dynamic Reference</td>
<td>204</td>
</tr>
<tr>
<td>General Tab</td>
<td>204</td>
</tr>
<tr>
<td>Input Tab</td>
<td>205</td>
</tr>
<tr>
<td>Set Exchange Variable</td>
<td>208</td>
</tr>
<tr>
<td>General Tab</td>
<td>208</td>
</tr>
<tr>
<td>Input Tab</td>
<td>209</td>
</tr>
<tr>
<td>Throw Fault</td>
<td>210</td>
</tr>
<tr>
<td>General Tab</td>
<td>210</td>
</tr>
<tr>
<td>Transform</td>
<td>211</td>
</tr>
<tr>
<td>General Tab</td>
<td>211</td>
</tr>
<tr>
<td>Input Tab</td>
<td>213</td>
</tr>
<tr>
<td>Output Tab</td>
<td>213</td>
</tr>
<tr>
<td>Validate XML</td>
<td>214</td>
</tr>
<tr>
<td>General Tab</td>
<td>214</td>
</tr>
<tr>
<td>Input Tab</td>
<td>216</td>
</tr>
<tr>
<td>Output Tab</td>
<td>216</td>
</tr>
<tr>
<td>XPath Route</td>
<td>218</td>
</tr>
<tr>
<td>General Tab</td>
<td>218</td>
</tr>
<tr>
<td>Decision Tab</td>
<td>218</td>
</tr>
</tbody>
</table>
## Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Mediation components</td>
<td>2</td>
</tr>
<tr>
<td>Figure 2</td>
<td>A mediation example</td>
<td>3</td>
</tr>
<tr>
<td>Figure 3</td>
<td>The Mediation Flow Editor</td>
<td>4</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Paths in a mediation flow for each message exchange pattern</td>
<td>7</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Mediation exchange information</td>
<td>10</td>
</tr>
<tr>
<td>Figure 6</td>
<td>The mediation flow Properties view</td>
<td>22</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Setting the exchange variable on the Input tab</td>
<td>39</td>
</tr>
<tr>
<td>Figure 8</td>
<td>The simplified schema editor dialog</td>
<td>40</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Relationship between a mediation flow and elements of a composite</td>
<td>45</td>
</tr>
<tr>
<td>Figure 10</td>
<td>The Input tab of an Invoke Operation task</td>
<td>51</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Configuring a Log task—the standard Log tab</td>
<td>56</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Configuring a Log task—the mapper Log tab for custom messages</td>
<td>56</td>
</tr>
<tr>
<td>Figure 13</td>
<td>An example using a Route task</td>
<td>60</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Output path for a route</td>
<td>62</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Fault path for a route</td>
<td>62</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Adding a routing case</td>
<td>65</td>
</tr>
<tr>
<td>Figure 17</td>
<td>Mapping values to variables</td>
<td>69</td>
</tr>
<tr>
<td>Figure 18</td>
<td>Routing conditions for the Route task</td>
<td>70</td>
</tr>
<tr>
<td>Figure 19</td>
<td>Routing with more than one variable</td>
<td>71</td>
</tr>
<tr>
<td>Figure 20</td>
<td>The Input tab of a Transform task</td>
<td>76</td>
</tr>
<tr>
<td>Figure 21</td>
<td>A travel reservation mediation flow: the input path</td>
<td>78</td>
</tr>
<tr>
<td>Figure 22</td>
<td>A basic mapping example</td>
<td>79</td>
</tr>
<tr>
<td>Figure 23</td>
<td>The XPath Editor window</td>
<td>80</td>
</tr>
<tr>
<td>Figure 24</td>
<td>Dragging a data element into a function</td>
<td>80</td>
</tr>
<tr>
<td>Figure 25</td>
<td>The completed transformation example</td>
<td>81</td>
</tr>
<tr>
<td>Figure 26</td>
<td>An example Choose statement</td>
<td>100</td>
</tr>
<tr>
<td>Figure 27</td>
<td>Example of If statement</td>
<td>101</td>
</tr>
<tr>
<td>Figure 28</td>
<td>Example of For Each statement</td>
<td>102</td>
</tr>
<tr>
<td>Figure</td>
<td>Description</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>29</td>
<td>An example of adding a variable to a mapping</td>
<td>103</td>
</tr>
<tr>
<td>30</td>
<td>Static and dynamic binding</td>
<td>122</td>
</tr>
<tr>
<td>31</td>
<td>Service providers and pass-through composites</td>
<td>123</td>
</tr>
<tr>
<td>32</td>
<td>Dynamic and static target interfaces</td>
<td>125</td>
</tr>
<tr>
<td>33</td>
<td>An example of pattern variables</td>
<td>126</td>
</tr>
<tr>
<td>34</td>
<td>Usage cases for Set Dynamic Reference task</td>
<td>128</td>
</tr>
<tr>
<td>35</td>
<td>Sending a reply message</td>
<td>137</td>
</tr>
<tr>
<td>36</td>
<td>Setting the reply message</td>
<td>138</td>
</tr>
<tr>
<td>37</td>
<td>Handling the reply message</td>
<td>139</td>
</tr>
<tr>
<td>38</td>
<td>Examples of where faults can occur</td>
<td>142</td>
</tr>
<tr>
<td>39</td>
<td>An example fault path</td>
<td>143</td>
</tr>
<tr>
<td>40</td>
<td>Example of configuring the Throw Fault task</td>
<td>146</td>
</tr>
<tr>
<td>41</td>
<td>Configuring a Catch Fault task</td>
<td>147</td>
</tr>
<tr>
<td>42</td>
<td>Removing specific faults from the target operation fault path</td>
<td>148</td>
</tr>
<tr>
<td>43</td>
<td>Catch Fault task for the mediation flow</td>
<td>149</td>
</tr>
</tbody>
</table>
## Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>General Typographical Conventions</td>
<td>xviii</td>
</tr>
<tr>
<td>Table 2</td>
<td>Syntax Typographical Conventions</td>
<td>xix</td>
</tr>
<tr>
<td>Table 3</td>
<td>Icons and Buttons used by Context Parameters</td>
<td>32</td>
</tr>
<tr>
<td>Table 4</td>
<td>Buttons for creating schema elements</td>
<td>40</td>
</tr>
<tr>
<td>Table 5</td>
<td>Icons for schema components</td>
<td>86</td>
</tr>
<tr>
<td>Table 6</td>
<td>Qualifier characters for schema components</td>
<td>87</td>
</tr>
<tr>
<td>Table 7</td>
<td>Schema for message properties and message flow context</td>
<td>88</td>
</tr>
<tr>
<td>Table 8</td>
<td>Data/Function tabs</td>
<td>91</td>
</tr>
<tr>
<td>Table 9</td>
<td>TIBCO XPath Functions</td>
<td>92</td>
</tr>
<tr>
<td>Table 10</td>
<td>Mapper toolbar buttons</td>
<td>97</td>
</tr>
<tr>
<td>Table 11</td>
<td>Right-click menu in the message panel</td>
<td>98</td>
</tr>
<tr>
<td>Table 12</td>
<td>Supported SQL types and their mapping to XML</td>
<td>117</td>
</tr>
<tr>
<td>Table 13</td>
<td>Custom mediation task plug-ins</td>
<td>155</td>
</tr>
<tr>
<td>Table 14</td>
<td>Icons for custom mediation tasks</td>
<td>156</td>
</tr>
<tr>
<td>Table 15</td>
<td>Runtime exception handling for custom mediation tasks</td>
<td>171</td>
</tr>
<tr>
<td>Table 16</td>
<td>Supported SQL types and their mapping to XML</td>
<td>192</td>
</tr>
<tr>
<td>Table 17</td>
<td>AutoMediate Command Options</td>
<td>228</td>
</tr>
<tr>
<td>Table 18</td>
<td>AutoMediate Command-Line Examples</td>
<td>229</td>
</tr>
<tr>
<td>Table 19</td>
<td>AutoMediate Command-Line Exception</td>
<td>230</td>
</tr>
<tr>
<td>Table 20</td>
<td>AutoMediate ANT Command-Line Examples</td>
<td>232</td>
</tr>
</tbody>
</table>
Preface

ActiveMatrix is a scalable and extensible platform for developing, deploying, and managing applications that conform to a service-oriented architecture. Mediation within ActiveMatrix allows your enterprise to easily adapt to changing requirements by providing an easy-to-use interface for managing mediation flows.

Topics

- Related Documentation, page xvi
- Typographical Conventions, page xviii
- Connecting with TIBCO Resources, page xxi
Related Documentation

This section lists documentation resources you may find useful.

TIBCO ActiveMatrix® Service Grid Documentation

The following documents form the TIBCO ActiveMatrix Service Grid documentation set:

- **Concepts**: Read this manual before reading any other manual in the documentation set. This manual describes terminology and concepts of ActiveMatrix. The other manuals in the documentation set assume you are familiar with the information in this manual.

- **Development Tutorials**: Read this manual for a step-by-step introduction to the process of developing and running composites.

- **Composite Development**: Read this manual to learn how to develop, debug, and package composites.

- **Java Component Development**: Read this manual to learn how to configure and implement Java components.

- **Mediation Component Development**: Read this manual to learn how to design mediation components.

- **Mediation API Reference**: Read this manual to learn about the mediation API.

- **Spring Component Development**: Read this manual to learn how to design Spring components.

- **Web Application Component Development**: Read this manual to learn how to design web application components.

- **Administration Tutorials**: Read this manual for a step-by-step introduction to the process of creating and starting the runtime, starting servers, and deploying applications to the runtime.

- **Administration**: Read this manual to learn how to manage the runtime, and deploy and manage services.

- **TIBCO ActiveMatrix Service Grid Installation**: Read this manual to learn how to install TIBCO ActiveMatrix Service Grid and configure and upgrade TIBCO ActiveMatrix Service Grid runtime objects.

- **TIBCO ActiveMatrix Service Grid Release Notes**: Read the release notes for a list of new and changed features. This manual also contains lists of known issues and closed issues for this release.
The documentation for the following TIBCO ActiveMatrix Service Grid components is installed with those components:

- TIBCO ActiveMatrix Implementation Type for C++
- TIBCO ActiveMatrix Binding Type for EJB
- TIBCO ActiveMatrix Binding Type for Adapters

Other TIBCO Product Documentation

You may find it useful to read the documentation for the following TIBCO products:

- TIBCO Enterprise Message Service™
## Typographical Conventions

The following typographical conventions are used in this manual.

### Table 1 General Typographical Conventions

<table>
<thead>
<tr>
<th>Convention</th>
<th>Use</th>
</tr>
</thead>
</table>
| TIBCO_HOME   | Many TIBCO products are installed within the same home directory. This directory is referenced in documentation as TIBCO_HOME. The value of TIBCO_HOME depends on the operating system. For example, on Windows systems the default value is C:\tibco. Other TIBCO products are installed into an installation environment. Products installed into different installation environments do not share components. Incompatible products and multiple instances of the same product must be installed into different installation environments. An installation environment consists of the following properties:  
- **Name** Identifies the installation environment. The name is a component of the path to the product in the Windows Start > All Programs menu. This directory is referenced in documentation as ENV_NAME.  
- **Directory** The directory into which the product is installed. This directory is referenced in documentation as TIBCO_HOME. The value of TIBCO_HOME depends on the operating system. For example, on Windows systems the default value is C:\Program Files\tibco\amx-3. |
| ENV_NAME     | A TIBCO configuration home stores configuration data generated by TIBCO products. Configuration data can include sample scripts, session data, configured binaries, logs, and so on. This folder is referenced in documentation as CONFIG_HOME. The default location of the folder is USER_HOME/ENV_NAME/data. For example, on Windows, the default location is C:\Documents and Settings\UserName\Application Data\ENV_NAME\data. |
| CONFIG_HOME  | Code font identifies commands, code examples, filenames, pathnames, and output displayed in a command window. For example:  
Use MyCommand to start the foo process. |
| code font    |                                                                                                                                 |

TIBCO ActiveMatrix Service Grid Mediation Component Development
**Table 1  General Typographical Conventions (Cont’d)**

<table>
<thead>
<tr>
<th>Convention</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>bold code font</strong></td>
<td>Bold code font is used in the following ways:</td>
</tr>
<tr>
<td></td>
<td>• In procedures, to indicate what a user types. For example: Type <em>admin</em>.</td>
</tr>
<tr>
<td></td>
<td>• In large code samples, to indicate the parts of the sample that are of particular interest.</td>
</tr>
<tr>
<td></td>
<td>• In command syntax, to indicate the default parameter for a command. For example, if no parameter is specified, <em>MyCommand</em> is enabled: *MyCommand [enable</td>
</tr>
<tr>
<td>italic font</td>
<td>Italic font is used in the following ways:</td>
</tr>
<tr>
<td></td>
<td>• To indicate a document title. For example: <em>See Concepts</em>.</td>
</tr>
<tr>
<td></td>
<td>• To introduce new terms. For example: A portal page may contain several portlets. <em>Portlets</em> are mini-applications that run in a portal.</td>
</tr>
<tr>
<td></td>
<td>• To indicate a variable in a command or code syntax that you must replace. For example: <em>MyCommand pathname</em></td>
</tr>
<tr>
<td>Key combinations</td>
<td>Key name separated by a plus sign indicate keys pressed simultaneously. For example: <em>Ctrl+C</em>.</td>
</tr>
<tr>
<td></td>
<td>Key names separated by a comma and space indicate keys pressed one after the other. For example: <em>Esc, Ctrl+Q</em>.</td>
</tr>
</tbody>
</table>

- The note icon indicates information that is of special interest or importance, for example, an additional action required only in certain circumstances.
- The tip icon indicates an idea that could be useful, for example, a way to apply the information provided in the current section to achieve a specific result.
- The warning icon indicates the potential for a damaging situation, for example, data loss or corruption if certain steps are taken or not taken.

**Table 2  Syntax Typographical Conventions**

<table>
<thead>
<tr>
<th>Convention</th>
<th>Use</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ ]</td>
<td>An optional item in a command or code syntax.</td>
</tr>
<tr>
<td></td>
<td>For example:</td>
</tr>
<tr>
<td></td>
<td><em>MyCommand [optional_parameter] required_parameter</em></td>
</tr>
<tr>
<td>Convention</td>
<td>Use</td>
</tr>
<tr>
<td>------------</td>
<td>-----</td>
</tr>
<tr>
<td></td>
<td>A logical 'OR' that separates multiple items of which only one may be chosen. For example, you can select only one of the following parameters: `MyCommand para1</td>
</tr>
<tr>
<td>{}</td>
<td>A logical group of items in a command. Other syntax notations may appear within each logical group. For example, the following command requires two parameters, which can be either the pair <code>param1</code> and <code>param2</code>, or the pair <code>param3</code> and <code>param4</code>. `MyCommand {param1 param2}</td>
</tr>
</tbody>
</table>
Connecting with TIBCO Resources

How to Join TIBCOmmunity

TIBCOmmunity is an online destination for TIBCO customers, partners, and resident experts, a place to share and access the collective experience of the TIBCO community. TIBCOmmunity offers forums, blogs, and access to a variety of resources. To register, go to http://www.tibcommunity.com.

How to Access All TIBCO Documentation

After you join TIBCOmmunity, you can access the documentation for all supported product versions here:

http://docs.tibco.com

How to Contact TIBCO Support

For comments or problems with this manual or the software it addresses, please contact TIBCO Support as follows.

- For an overview of TIBCO Support, and information about getting started with TIBCO Support, visit this site:
  http://www.tibco.com/services/support

- If you already have a valid maintenance or support contract, visit this site:
  https://support.tibco.com

Entry to this site requires a user name and password. If you do not have a user name, you can request one.
Chapter 1  Introduction

ActiveMatrix provides service mediation to help your enterprise adapt to a rapidly changing environment. This chapter describes service mediation.

Topics

- Overview of Mediation, page 2
- Mediation Flows, page 4
- Designing Mediation Flows, page 12
Overview of Mediation

Mediation is part of a Service Oriented Architecture (SOA) for applications. *Mediation* involves virtualizing and managing service interactions between communicating participants. ActiveMatrix implements a component-based platform to implement SOA within an enterprise.

A *mediation component* shields service consumers from the service provider’s physical location from both the design-time and run-time perspective. A mediation component is responsible for delivering requests to a service provider. Service providers can respond to requests delivered by mediation components without needing to know the point of origin of the message.

*Figure 1  Mediation components*

Mediation components provide a mechanism for handling changing service requirements. Mediation components enable you to provide functionality that uses and enhances existing services, without changing the underlying service description.
This example shows a mediation component that provides approvals for loan applications. The existing loan service might require credit scores, loan amount, employment history, and so on. To enable quick turnaround for smaller loans, you might want to provide a new service that approves loan requests for under $50,000 for all applicants with credit scores above 700.

Instead of rewriting your existing service to handle new types of requests, a mediation component can accept requests that contain information from the loan requestor and then submit the request to the appropriate service for approval.

Using the Mediation Flow Editor, you can easily create mediation components that operate within the SOA-based ActiveMatrix platform.
Mediation Flows

A mediation flow is a graphical representation of the business logic for a mediation component. Mediation flows are created and managed within the Mediation Flow Editor, in TIBCO Business Studio. Mediation flow resources are stored in the Mediation Flows folder within a ActiveMatrix SOA project.

See the Composite Development for more information about creating and managing ActiveMatrix SOA Projects.

Figure 3  The Mediation Flow Editor

A mediation flow includes several parts:

- **Mediation interfaces**  One or more mediation interfaces provide the interface for the mediation components that you expose to consumers of your applications.

- **Mediation paths**  Depending on the message exchange pattern of the mediation interface, there can be an input, output, and fault path from each operation in the mediation interface to operations in the target interfaces.

- **Target interfaces**  These interfaces to existing services in your enterprise provide implementation of the operations for the associated mediation operations.

- **Mediation tasks**  You can place mediation tasks, such as Log or Route, on mediation paths to perform business logic your application requires.
Message Exchange Patterns

Mediation flows support two web service message exchange patterns (MEP) for mediation and target operations:

- **One-way (in-only)** A message consumer sends a message to a provider.
  In this exchange, the mediation flow allows only an input path from the mediation operation. No output path is used. Fault paths exist to handle any errors produced by mediation tasks executing on the input path. To terminate the mediation of a one-way operation without invoking a target operation, use the End Mediation task. (See End Mediation on page 178 for more information.)

- **Request-response (in-out)** A message consumer sends a message to a provider, and the provider sends a response message back to the consumer.
  In this exchange, the mediation flow has three paths:
  — An input path for the message from the consumer to the provider
  — An output path for the reply message
  — A fault path for any faults that are encountered during processing.

To mediate different operations with a target operation of a different message exchange pattern, use the Invoke Operation and Generate Reply mediation tasks. See Generate Reply on page 180 and Invoke Operation on page 182 for more information.

Mediation Flow Interfaces

Web Service Description Language (WSDL) files define the interface to a web service. WSDL is a standard maintained by the World Wide Web Consortium; it is beyond the scope of this guide to describe WSDL syntax and functionality in detail. You can learn more about WSDL from http://www.w3.org/TR/wsdl; commercial publications about Web Services and WSDL files are also available.

Mediation flows have two types of interfaces:

- **Target interfaces** are interfaces to the actual services that make up your enterprise application. These interfaces appear on the right side of a mediation flow in the Mediation Flow Editor.

- **Mediation interfaces** are interfaces that you expose to the consumers of your services. Mediation interfaces can have the same number and type of operations as target operations, or they can be different from the target interfaces. Mediation interfaces appear on the left side of a mediation flow in the Mediation Flow Editor.
See Figure 3, The Mediation Flow Editor, on page 4.

Interfaces (also called *port types*) in mediation flows are references to abstract web services that a WSDL file defines—interfaces in a mediation flow do not have concrete bindings. The WSDL files you use in a mediation flow might have concrete bindings, but the mediation flow is concerned only with receiving the message from the mediation operation, processing the message, and forwarding it to its target operation. Binding occurs when a mediation flow is placed into an ActiveMatrix mediation component, using the Composite Editor. See *Mediation Components on page 45* for more information about components.

You can add the same interface more than once to either the mediation or target interface list. Adding the same interface several times to the mediation interfaces list enables you to offer the same interface to consumers with different mediation implementations.

This functionality can be used to offer different qualities of service to different consumers of the service. Adding the same interface several times to the target interfaces list enables you to bind the same interface to different providers, if you have more than one provider of the same service.

### Planning Target and Mediation Interfaces

Designing a mediation flow requires that you plan how your services will be exposed by mediation interfaces. You might have a one-to-one mapping of target and mediation interfaces, or you might expose mediation interfaces that are very different from your target interfaces. The requirements of your application will drive the design of your mediation flows.

For example, you might have target services that are offered from a third party and therefore cannot change the target interfaces. In this case, if you want to offer a service that uses the third-party services but has different operations and message schemas, you must create your own interface/WSDL file describing the service to offer, and use that interface file as the mediation interface.

Before starting your project, consider the requirements of your application, plan and develop the required interfaces, and determine how the mediation interfaces will use the target interfaces.

### Paths in a Mediation Flow

Each incoming message for an operation in a mediation interface follows an input path to a target operation, or a task that terminates the input path. Depending on the message exchange pattern, there could also be an output path for reply messages and a fault path for fault messages.
For operations that use the one-way message exchange pattern, there is only an input path from the mediation operation to the target operation. Operations that use the request-response message exchange pattern have an input path, an output path, and a fault path. Fault paths handle faults wherever they occur in a mediation flow—either during processing within the mediation flow, or during processing by the target operation.

The Mediation Flow Editor enables you to view the input, output, and fault paths for an operation by selecting the mediation operation, and then clicking on the appropriate button in the mediation paths area of the editor. Only the path for the currently selected mediation operation appears in the mediation paths area.

When the input path for a mediation operation is defined or changed, the output and fault paths are automatically changed to reflect the input path. Output or fault messages must be returned to the original invoker, so that the input, output, and fault paths are automatically kept compatible.

You can use Route tasks to divide a mediation path into multiple sub-paths to potential target operations. Route tasks allow the mediation path to be split into multiple sub-paths to potential target operations. While the path shows multiple potential destinations, each message is only sent to one destination. The path in the Mediation Flow Editor is like a map that describes the potential places where a message can go. When the mediation flow is executed, however, each message travels to only one target operation.
You can use multiple, nested route tasks to send a single message to a target in several different ways. Rather than using a single route task with compound conditions, the use of nested routes enables you to make complex routing decisions that are easier to follow.

Mediation Tasks

You can place mediation tasks on input, output, or fault paths, to perform business logic required by your application. For example, if the schema of the input message of your mediation operation does not match the schema of the input message of the target operation, you can use a Transform task to change the schema to the desired format.

The Mediation Flow Editor includes a variety of mediation tasks:

- **Invoke Operation** Enables you to invoke an operation of an interface in the target interface list during processing of an input, output, or fault path. For example, you can invoke an operation on the input mediation path and use the data in the reply message in subsequent tasks in the input path before the mediation flow invokes the specified target operation. See Chapter 3, Invoking an Operation, on page 49.

- **Query Database** Performs a SQL SELECT statement on a database. The task can specify one or more tables in the FROM clause of the SELECT statement, one or more columns to return in the SELECT list, and one or more conditions in the WHERE clause. Optionally, you can specify the maximum number of rows to return. See Chapter 7, Querying a Database, on page 111.

- **Log** Writes information to the log file. You can use this task for auditing, security, or other purposes. See Chapter 4, Logging Mediation Exchange Information, on page 53.

- **Transform** Takes information from the mediation exchange (described in Mediation Exchange on page 10) and changes it to the appropriate format. See Chapter 6, Transforming Data in a Mediation Exchange, on page 75.

- **Parse XML** Used when you have an XML document stored in a string or binary field. This task produces a tree representation of the XML that can be used by subsequent tasks in the mediation flow. This task can be paired with the Render XML task to convert the parsed XML back into a string or binary field for transmission within a message. See Parse XML on page 188.

- **Render XML** Converts an XML tree for a specified schema to a string or binary element that contains the XML document. This task can be paired with the Parse XML task to convert the parsed XML back into a string or binary field for transmission within a message. See Render XML on page 195.
- **Validate XML** Provides validation of XML messages using an XML Schema that is configured at design time or specified dynamically at runtime. Validation errors may be caught and handled in the current flow path, or handled by the fault catch mechanism. See Validate XML on page 214 for more information.

- **Set Context** Provides a way to set HTTP header values or JMS user property values of the operations within a mediation flow. See Working with Message Context Properties on page 30 and Set Context on page 202 for more information.

- **Set Exchange Variable** Sets the value of the items within the exchange variable. The **Input** tab of the Set Exchange Variable task is a mapper panel that enables you to set the exchange variable for the currently selected operation. See Setting the Exchange Variable on page 38, and Set Exchange Variable on page 208.

- **Route and XPath Route** Route tasks enable you to specify more than one potential destination for messages sent by a mediation operation. The message is sent to the appropriate target operation based on criteria you specify. In Route tasks, the criteria for routing conditions are simple comparison operations. XPath Route tasks are similar to Route tasks, but you can specify more complex criteria for routing conditions. See Chapter 5, Routing Messages in a Mediation Flow, on page 59 for more information on Route tasks.

- **Throw Fault** Stops processing in the current mediation flow and transfers control to the fault path. This task is useful if a mediation operation is deprecated and you want to return a fault to the requestors of the operation. This task is also useful if you want to specify fault conditions for Route or XPath route tasks. See Chapter 10, Fault Processing in a Mediation Flow, on page 141 for more information about faults and the Throw Fault task.

- **Generate Reply and Handle Reply** In some situations, you might want to send a reply message to a consumer without invoking the target operation. The Generate Reply and Handle Reply tasks enable you to bypass the target operation and send reply messages to the consumer directly from the mediation flow. See Chapter 9, Replying to Messages, on page 135 for more information on Generate Reply and Handle Reply tasks.

- **End Mediation** Ends a one-way (in-only) message exchange pattern for operations that don’t involve a response. End Mediation includes a message re-delivery feature, so that you can request that a message be re-delivered if it encounters a fault during processing.

- **Set Dynamic Reference** Provides the values needed for resolving a service provider in a dynamic target interface. Each Set Dynamic Reference task sets the value of the service provider for the specified dynamic target interface.
The value is then used by the next service invocation that refers to that
dynamic target interface. See Setting the Dynamic Reference Task on page 127

You can extend the functionality of mediation flows by creating your own
Custom tasks. See Chapter 11, Custom Mediation Tasks, on page 151.

Mediation Exchange

When a mediation operation receives a message, a mediation exchange is created
to hold information related to the message and the mediation flow. Information in
the mediation exchange is available to tasks in the mediation flow.

Figure 5  Mediation exchange information

The mediation exchange consists of this information:

- **Mediation flow properties** You can define properties on a mediation flow to
  store information used within the flow. For example, you might create a
  property to store currency exchange rates, or calendar holidays for system
down time.

- **Mediation flow context** Includes information such as component name and
  mediation flow information, if the Mediation Flow Context option is set on the
  Advanced tab of the mediation operation’s Properties view. See Working with
  Message Context Properties on page 30 for more information.
- **Message context**  The context of the message sent to the mediation operation. Message context includes information about the message transport (for example, HTTP or JMS message headers) and security context information about the message. You can use the Set Context task to set HTTP header values and JMS user property values within a mediation flow—see Working with Message Context Properties on page 30 and Set Context on page 202.

- **Message data**  Content of the message. The content of this item depends on the processing within a mediation flow. For example, for input paths this component contains the schema of the input message of the mediation or the target operation. For output paths, this component contains the schema of the reply message of the mediation or the target operation. Similarly, for fault paths this component contains the schema of the fault message.

  Some mediation tasks, such as Transform, can change the contents of the message data.

- **Exchange variable**  A defined schema to hold data that persists through all paths of a mediation operation (input, output, and fault paths). You can use any schema stored in the project to define the structure of the exchange variable. The value of the variable is set during execution of the mediation path with the Set Exchange Variable task. See Working with Exchange Variables on page 37 and Set Exchange Variable for more information.

- **Contributed data**  Mediation tasks, such as the Transform task or a custom mediation task, can add—contribute—data to the mediation exchange. When the data is added, subsequent tasks can access each task’s added data. An option on some mediation tasks enables you to specify whether you want the task to change the existing message data in the mediation exchange, or place the results of the task into a new data item in the mediation exchange.
Designing Mediation Flows

You can design mediation flows from the top down, or from the bottom up. That is, you can start with interfaces and mediation flows, or you can start by designing composites and components.

If you start with interfaces, you can create mediation flows from the interfaces. If you start with components, you can assign a mediation flow as the implementation of the component after specifying the services and references in the Composite editor.

Designing a mediation flow generally follows these steps:

1. Create an ActiveMatrix SOA project and import the WSDL files.
2. Virtualize interfaces.
   a. Create a mediation flow.
   b. Specify the mediation and target interfaces.
   c. Create mediation paths.
3. Select a mediation patterns.
   The most cited ESB (mediation) patterns are these:
   — VETO (Validate, Enhance, Transform, Operate)
   — VETRO (Validate, Enrich, Transform, Route, Operate)
   Mediation provides the Validate, Transform and Route tasks.
   The Enrich task can be achieved using the Query DB task, the Invoke Operation task, or a customer-created task.
   The Operate task makes the target service call.
   a. Add and remove tasks.
   b. Configure task properties.
   c. Handle faults.
5. Bind and deploy.
   a. Create composite and components.
   b. Specify bindings.
   c. Assemble and run.

Before you package and deploy your project, ensure that all validation errors are resolved. An error icon \( \times \) appears on the operation name of mediation interfaces with errors.

Errors occur because of an invalid configuration. Each error is logged on the Problems tab of the mediation flow.

For more information about the process of designing a mediation flow, consult these resources:

- *Composite Development* describes the first step in the process, creating the project and obtain the interfaces.
- Chapter 2, Working with Mediation Flows, on page 15 describes steps 2 through step 4a in more detail.
- *Composite Development* describes how to create service assemblies for deployment and execution.
- *Administration* describes how to deploy and run your project.
Chapter 2  Working with Mediation Flows

This chapter describes how to create and manipulate mediation flows and how to work with mediation components in the Component Editor.

Topics

- Creating Mediation Flows, page 16
- Mediation Flow Properties, page 22
- Working with Interfaces, page 24
- Working with Paths, page 28
- Working with Message Context Properties, page 30
- Working with Exchange Variables, page 37
- Working with Tasks, page 42
- Mediation Components, page 45
Creating Mediation Flows

You use the mediation flow wizard to create new mediation flows and mediation flows from existing web services.

Before creating mediation flows, you should have at least one WSDL file that defines the target interface that you plan to mediate.

For more information about folders in ActiveMatrix SOA projects, see the Installation.

Starting the Mediation Flow Wizard

To start the mediation flow wizard:

1. Right-click the Mediation Flows folder in the Project Explorer.
2. Choose New > Mediation Flow from the pop-up menu.
   The Create Mediation Flow dialog opens.

   The default option is to create an empty mediation flow.
Creating a New, Empty Mediation Flow

Creating a new, empty mediation flow enables you to start a mediation flow from scratch. You should at least have one WSDL file that describes the interface that you plan to mediate, but you can have zero or more target interfaces.

To create a new empty mediation flow:

1. Start the mediation flow wizard.
   - Right-click the Mediation Flows folder in the Project Explorer.
   - Choose New > Mediation Flow from the pop-up menu.

   The Create Mediation Flow dialog opens. The default option to create an empty mediation flow is active.

   You can also select the Create Composite checkbox if you want to create a corresponding composite for this mediation flow. See Mediation Components on page 45 for more information about working with components and composites.

2. Click Next.

3. Supply a name in the Mediation Flow Name field.
   - If you chose to create a composite to correspond to the mediation flow, you can also name the composite in the Composite Name field.

   You can also specify a different folder in the project for the mediation flow (and composite, if one is created).

4. Click Finish.

   The Mediation Flow Editor opens. You can begin to add interfaces and configure your mediation flow.

Creating New Mediation Flows From Existing Web Services

If you have existing web services that you want to mediate, you can create new mediation flows for each interface. This is useful if you have many services and you want to create one mediation flow for each service.

Before creating mediation flows, ensure that the WSDL files that describe the interfaces have been imported into the project. See Composite Development for more information about importing WSDL files.

To create mediation flows from existing services:
1. Start the mediation flow wizard.
   a. Right-click the Mediation Flows folder in the Project Explorer.
   b. Choose **New > Mediation Flow** from the pop-up menu.
2. Select the option **Multiple Mediation Flows Using Existing Web Services**.
   You can also select the Create Composite checkbox if you want to create a corresponding composite for the mediation flows. See **Mediation Components on page 45** for more information about working with components and composites.
3. Click **Next** to select the interfaces for the mediation flow.

   ![Create Mediation Flow](image)

4. Select the WSDL files to use when you create mediation flows.
   You can select and add WSDL files individually, or you click the **Add All>>** button to add all files to the Selected Interfaces list. You can remove one or more WSDL files using the **<Remove** and **<<Remove All** buttons.
   A mediation flow is created for each WSDL file. The target interfaces and mediation interfaces are the same, and a path is automatically created between operations of the same name. If a WSDL file includes more than one port type, each port type is added to the mediation flow created for the file.
   The name of each mediation flow is based on the name of the WSDL file. If there are conflicts in mediation flow names, a numbering scheme ensures that each newly created mediation flow has a unique name.
5. The next step depends on whether you checked the Create Composite option:
   a. If you did not check the Create Composite option, click **Finish** to create the mediation flows and composite.
   b. If you checked the option to create a composite, click **Next** to specify the binding type— JMS, SOAP/HTTP, SOAP/JMS.

   If you select SOAP/HTTP, additionally specify the **Connector**.

   Click **Finish** to create the mediation flows and composite.

---

**Editing Preferences of the Mediation Flow Editor**

You can set preferences for the Mediation Flow Editor:

- Mediation flow
- Task icon size
- Tree expansion level
- Query database task validation

To change the preferences:
1. Select **Window > Preferences** to open the Preferences dialog:

   ![Preferences dialog](image)

   1. Set values in the **Mediation Flow** section:
      a. In the **Mediation Flow Name in Wizard** text box, provide the default name of mediation flows that you create with the **Single Empty Mediation Flow** option in the wizard.
      b. In the **Mediation Flow Folder Name**, provide the name of the folder in which to store mediation flows.
      c. Clear the **Enable Diagram Tooltips** check box if you want to disable the tooltips.
      d. Clear the **Enable Connection Animation** check box if you want to disable animation.

   2. Set values in the **Mediation Task Icon Size** section:
      a. Check the **Small Icons (16x16)** radio button to display small icons in mediation flows and the palette.
      b. Check the **Large Icons (32x32)** radio button to display icons in mediation flows and the palette.

   3. In the **Mediation Task Mapper Tree Expansion Level** text box, type the default value for the depth you want to expand the left and right sides of the mapper.

   4. Set values in the **Live Database Validation For Query Database Task** section:
      a. Check the **Validate Query Database Task Data** checkbox to connect to the database during validation to determine if the Query Database task
configuration is correct. The information being queried is the structure (tables and columns) of the database.

b. In the **Validation Timeout** field, provide the timeout (in seconds) for the validation task.

c. Check the **Reload Database Data When Validation** checkbox, if the database structure is changing, to query the database each time the validation process is run.
Mediation Flow Properties

You can define mediation flow properties to store information. For example, you might want to store values for current price markups, currency rates, or user names. The properties you create are stored in the mediation exchange, and tasks in a mediation flow can use them.

Properties are defined and removed using the Properties view of the mediation flow.

Figure 6 The mediation flow Properties view

To add a mediation flow property:
1. Click the ellipsis (…) in the Name field, or click the Add button to the right of the table, and
2. Specify a name, data type, and value for the property.

You can select one of four property types:
- String
- Integer
- Boolean
- JDBC Resource Template

You can also specify properties when you create a mediation component. Component-level properties override the values of properties with the same name specified on the mediation flow.

To delete a property:
1. Select the property row in the table.
2. Press the Delete key or click the Delete button to the right of the table.
Validation

When a mediation flow is created, a property called `VALIDATE_MESSAGE_DATA` is added by default.

- At design time, a property `VALIDATE_MESSAGE_DATA` is defined in the mediation flow implementation. When set to `true`, this property is used to validate the incoming message. This property is accessed by the mediation component in the composite is exposed at the composite level as `MEDIATE_VALIDATE_MESSAGE_DATA`.

  Having this property at the mediation component level allows for fine-grained control compared to defining it at the mediation implementation level.

- A property `VALIDATE_MESSAGE_DATA` is available at the mediation implementation level when viewed using the Administrator UI. The default value of this property is `false`. Set this property to `true` to enable validation of message data received by the mediation component. Validation of message data happens for both the component service and reference.

  Setting this property to `true` at the mediation implementation level enables validation on every mediation component on that particular node.

The value of the property set at the mediation component level takes precedence over the value set at the mediation implementation level. At runtime, when the incoming message (either a request message on the mediation interface or a reply message on the target interface) into mediation fails validation, an undeclared fault is returned to the consumer. The fault message will only indicate that a validation failure has occurred with no details provided. For more details about the cause of the validation failure will be contained in the log files.
## Working with Interfaces

Interfaces are collections of operations that WSDL files define. WSDL files are typically contained in the Service Descriptors special folder in a project. You can obtain interfaces in a variety of ways, usually by importing WSDL files into a project or by using a UDDI registry service.

*Composite Development* describes the folders in an ActiveMatrix SOA project and how to obtain WSDL files and use UDDI registry services.

### Adding Interfaces to Mediation Flows

Before you add interfaces to your mediation flow, plan the needs of your application and determine which target and mediation interfaces you need. Some planning considerations are discussed in Planning Target and Mediation Interfaces on page 6.

Of the variety of ways to add interfaces to a mediation flow, the method you use depends upon the requirements of your application. If you plan to have a one-to-one relationship between target and mediation interfaces, you might use the technique described in Creating New Mediation Flows From Existing Web Services on page 17.

If a WSDL file contains more than one port type, you can expand the WSDL file in the project tree and select only the interface you want to drag and drop into the mediation flow. You can also drag and drop the top-level WSDL file to add all interfaces within the WSDL file to the mediation flow.

If your target interfaces and mediation interfaces have different operations and schemas, drag and drop each interface from the Project Explorer into the appropriate area of the mediation flow.

Dragging and dropping an interface onto the mediation side of a flow creates an untargeted flow for each operation. When you select the mediation operation for an untargeted flow, the flow appears as a line ending in a question mark. You can then use a Generate Reply, Throw Fault, or End Mediation task on the flow without having to add a target interface. You can also drag and drop an untargeted flow to a target interface and mediate that interface.
Another way to add an interface to the target or mediation interface list is to use the menu icon at the top of the Mediation Interfaces area and Target Interfaces area of the mediation flow. The menu contains an Add Mediation Interface or Add Target Endpoint item, depending upon which side of the mediation flow you use. The Add menu opens a Select WSDL Port Type dialog where you can choose an interface to add.

You can add more than one copy of the same interface to the mediation interfaces side of the mediation flow. Doing so allows you to specify different business logic for the same interface. You can then expose each implementation of the interface to different clients.

For example, you could use this functionality to offer different qualities of service to different clients.

**AutoMediate™**

The Mediation Flow Editor provides mechanisms for quickly adding identical interfaces to both the target and mediation interface sides of the mediation flow with corresponding mediation paths between operations of the same name. These mechanisms are known as AutoMediate.

You can use AutoMediate in these ways:

- Drag and drop an interface to either the target or mediation interface area. Select the interface you have added to the mediation flow, then click and drag it to the opposite side of the mediation flow.
- Drag and drop an interface to the target interface area. Then, click the projection icon in the title bar of the interface.
- Drag and drop an interface onto the mediation paths area (the center area) of a mediation flow.

All mediation operations are connected to their corresponding target operations.

You can use the TIBCO AutoMediate Command Line tool to use existing services as input to create a fully functional composite application that generates a DAA that you can deploy into an ActiveMatrix runtime environment. See TIBCO AutoMediate Command-Line Tool on page 223 for detailed information.
Deleting Interfaces

You can delete interfaces from either side of the mediation flow by clicking the interface menu icon in the title bar of each interface and selecting Delete from the pop-up menu.

Deleting a target interface that has one or more operations for which there are paths from mediation operations results in undefined destinations for the paths. You must define the destinations by directing them to other operations or to Throw Fault tasks.

Moving Interfaces

To move interfaces within the target and mediation interface list:
1. Click the header of the interface you want to move.
2. Drag the interface to the new location in the list of interfaces.

Creating Local WSDL Files

If you automatically create mediation interfaces, you might want to create local copies of the WSDL files. Creating local copies enables you to make changes to the copies without affecting other services or clients that might use those WSDL files.

To create a local copy of the WSDL file for your mediation interfaces:
1. In your mediation flow, locate the mediation interface you want to include in the local WSDL file.
2. Click the menu icon in the title bar of the interface and choose Copy Interface from the pop-up menu.
   
   The Mediation Flow Editor creates a local WSDL file and places it in the same folder as the mediation flow. The name of the file is the same as the name of the mediation flow, with the file extension .wsdl.
3. (Optional) Include additional mediation interfaces in the local WSDL file by repeating step 2 for each interface.

   Each interface you copy is placed into the same local WSDL file so that you can edit the file using the standard WSDL editor.

Policies

Mediation interfaces support the following intents:

- At least once
Specifies that the provider must receive every message sent to it by consumers at least once.

- Transacted one way
  Specifies that references must send all out-only messages within a global transaction, and the ActiveMatrix framework must deliver the message only after the transaction commits.

See *Composite Development* for more information on intents and policies.
# Working with Paths

Paths are created by dragging and dropping a mediation operation onto a target operation. Paths can be automatically created in mediation flows, as described in [Adding Interfaces to Mediation Flows on page 24](#).

Creating an input path also creates corresponding output and fault paths. You can click the Input, Output, and Fault icons at the top of the mediation paths area to view the corresponding path for each mediation operation.

The path for only one mediation operation appears in the mediation paths area. Select a mediation operation to view its path.

Mediation operations must have an input path. Typically the input path leads to a target operation, but there can also be a route task that splits the path into more than one destination, or the path can lead to one of these tasks:

- A Throw Fault task,
- A Generate Reply task
- An End Mediation task

If a mediation operation is not implemented—that is, if it does not have an input path—an error icon appears in the bottom left corner of the operation’s icon. You must implement all mediation operations in a mediation flow before deploying the project.

## Modifying Paths

On the input path, a small circle appears next to the directional arrow of the path. This allows you to change a path.

To modify the target operation:

<table>
<thead>
<tr>
<th>Goal</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Move path to a different target operation</td>
<td>Click the circle and drag it to a new location.</td>
</tr>
<tr>
<td>Move target operation on an output path</td>
<td>Click and drag the circle next to the target operation.</td>
</tr>
</tbody>
</table>

The input and fault paths are automatically updated.

To delete a path:
1. Select the path.

2. Press the **Delete** key, or right-click while hovering over the path and choose **Delete** from the pop-up menu.

Route tasks create sub-paths and have some additional characteristics. See Chapter 5, *Routing Messages in a Mediation Flow*, on page 59 for more information about working with route tasks.
Working with Message Context Properties

In a mediation exchange, the context of the message sent to the mediation operation includes information about the message:

- Information about the message transport (for example, HTTP headers or JMS message headers/properties)
- Security information about the message.

ActiveMatrix provides a way for the mediation flow to receive message context information and access its values in the mediation path. It also provides a mechanism for setting the message context data for the input message of the target operation and the output message of the mediation operation.

The Mediation Flow also allows you to configure an additional type of context parameter called Mediation. This type provides the security context, endpoint reference, and request message mechanisms.

Adding Context Parameters

Context parameters can be configured at a mediation interface level, target interface level, or for an operation contained in the interfaces. Parameters added at the interface level are available for use by all the containing operations. Parameters added at the operation level can be used by those operations only.

Context parameters for the Mediation and Target interfaces are independent of each other. The Set Context mediation task is used to map values of the defined context parameters.

To add context parameters:

1. Choose the interface or operation.
2. Select the General tab from the Properties view.
3. Click the button located on the right side of the table.
4. Specify the parameter properties:
   — Name: Name of the parameter.
   — Direction: Choose between Input, Output, and Fault.
   — Type: Choose between Basic, Message, Bag, Mediation.
   — Definition: This is the definition mechanism.

   If you chose Mediation as the context parameter type, the available definitions are Security Context, Endpoint Reference, and Request Message Context.

   If the Direction is Fault and the Type is Mediation, the Definition mechanism is set to Undeclared Fault Context. See Undeclared Fault Headers, page 35 for more information.

The context parameter is added to the chosen interface or operation.

**Deleting Context Parameters**

To delete context parameters from the Mediation or Target interface, choose the context parameter and click the button.

Context parameters can be deleted at either the interface level or the operation level.

If the context parameter is deleted at the interface level, the parameter is deleted from all the operations.

If the context parameter is deleted from the operation level, the parameter is deleted from that operation only. If that operation was the last operation to which the context parameter was applied it is then removed from the interface as well.

**Defining the Scope of Context Parameters**

Parameters added at the interface level are available for use by all the containing operations. Parameters added at the operation level are only available for that specific operation.

A parameter defined for operation A can be made available to operation B by selecting the parameter in operation B and clicking the button.

A parameter defined for an operation can be made available to all other operations by selecting the parameter at the interface level and clicking the button.
The scope of a parameter used by operations A and B can be reduced by selecting the parameter in the operation where the parameter will not be used and clicking the button. The operation will remain in the table but the tooltip will display the reduced scope.

**Table 3  Icons and Buttons used by Context Parameters**

<table>
<thead>
<tr>
<th>Icon/Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Icon" /></td>
<td>Indicates that the parameter is used by all operations in the interface.</td>
</tr>
<tr>
<td><img src="image2" alt="Icon" /></td>
<td>Indicates that the parameter is not used by all operations in the interface. The tooltip will list all operations in the interface that use this parameter.</td>
</tr>
<tr>
<td><img src="image3" alt="Icon" /></td>
<td>Indicates that the parameter is used in the currently selected operation. The tooltip will list all operations in the interface that use this parameter.</td>
</tr>
<tr>
<td><img src="image4" alt="Icon" /></td>
<td>Marks the selected parameter for use in the currently selected interface or operation.</td>
</tr>
<tr>
<td><img src="image5" alt="Icon" /></td>
<td>Add a parameter to the currently selected interface or operation.</td>
</tr>
<tr>
<td><img src="image6" alt="Icon" /></td>
<td>Removes the currently selected parameter from the operation or deletes it from the interface.</td>
</tr>
</tbody>
</table>
Working with Context Parameters

The context parameters available on the left side of the input mapper are contained under the root element `MessageContext`. Context parameters can be used like other elements in the tree.

The following figure shows the mapping of a context parameter, `userID`.

![Mapping of a context parameter](image)

The Set Context mediation task is used to set values for the context parameters. The input mapper for the Set Context task shows the context parameters defined for the target interface or the mediation interface if the Set Context task is on the output or fault path.

The following figure shows the mapping of two context parameters. The parameters `userID` and `userVal` are defined for mediation interfaces, and the parameters `ID` and `Val` are defined the target interface.

![Mapping of two context parameters](image)

**Mediation Context Parameters**

The MediationFlow also allows you to configure an additional type of context parameter called **Mediation**. This type provides functionality for security context. These parameters are available for the input direction only.
Security Context

The mediation component in the Composite Editor do not use these parameters. The values for these parameters are automatically passed to the mediation component and require no additional configuration.

When a parameter of type Mediation and definition mechanism Security Context is added to the interface, the security context and the SAML assertion data is available for security context based routing, transformations, or to log security context data.

The following figure show a simple mapping of the SAML assertion data:

![Image of SAML assertion data mapping](image)

Endpoint Reference

A parameter of type Mediation and definition mechanism Endpoint Reference provides access to the endpoint reference schema for the mediation interface.
The Set Dynamic Reference task is used to provide the endpoint reference to the target invocation. The following screen shows one such mapping between the context parameters of the service and the reference.

The Endpoint Reference Mechanism of the Set Dynamic Reference task is set to WS-A Endpoint Reference. See Set Dynamic Reference, page 204 for more information.

**Request Message Context**

Provides access to the CorrelationID, ContextID, and ParentContextID parameters.

**Undeclared Fault Headers**

A context parameter of type Mediation and direction Fault is used to access the undeclared headers (code, role, and reason) provided by SOAP. Using this type of context parameter requires the target interface to be bound to a SOAP endpoint, but no validation can be done to ensure that. The developer of the system just has to know that SOAP is being used.
The following screen shows the mapping between the context parameters on the Fault path of the Target interface and that of the Mediation interface.

**Using Context Parameters in Mediation Components**

Context parameters added to the interface or operation are propagated to the mediation component in the Composite Editor. The **General** tab of the Component Service and the Component Reference element has a section for context parameters. All context parameters defined in the mediation flow will be exposed in the mediation component, except for the context parameters of type **Mediation**.

Context parameters can be added to the Component Service or Component Reference. These context parameters can then be pushed down to the implementation level. Context parameters of type **Mediation** cannot be added to the Component Service or Component Reference.
Working with Exchange Variables

You can define an exchange variable for each mediation operation in your mediation flow. An exchange variable provides a location that stores data for use in all paths for a particular mediation operation.

For example, you might want to store a field from an incoming message, such as a correlation ID in a JMS header for the message. After it is stored, this data is available for all tasks in the input, output, or fault paths of a mediation operation.

Each mediation operation has one exchange variable. The exchange variable can have any structure. For example, the exchange variable can have repeating elements, if it is necessary to hold multiple instances of the same element.

After it is defined, the exchange variable is available for all tasks that can access the mediation exchange in the input, output, and fault paths of your mediation flow. The values of the fields of the exchange variable are empty until they are set using the Set Exchange Variable mediation task. Setting the Exchange Variable on page 38 describes this task.

Defining Exchange Variables

Exchange variables are defined in the Advanced section of the properties view of a mediation operation.

To define an exchange variable:

1. Select a mediation operation in the mediation editor.
2. Expand the Advanced option from the General tab.
Set **Exchange Variable supports only XSD elements**, so the schema definition for the exchange variable must be stored in an XSD within your workspace.

**To create an XSD (XML schema definition):**

<table>
<thead>
<tr>
<th>Starting Point</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Browse</strong> button</td>
<td>Use the simple XSD editor with the <strong>Create</strong> button.</td>
</tr>
<tr>
<td></td>
<td>See <strong>Creating Simple Schemas on page 39</strong> for more information about the Simplified Schema Editor that opens when you click the <strong>Create</strong> button.</td>
</tr>
<tr>
<td>TIBCO Business Studio</td>
<td>Use the XSD editor.</td>
</tr>
<tr>
<td></td>
<td>See the Eclipse documentation, <em>XSD Developer Guide</em>, for more information about the XSD editor in TIBCO Business Studio.</td>
</tr>
<tr>
<td>XSD editor plug-in</td>
<td>Use your own plug-in.</td>
</tr>
</tbody>
</table>

**Setting the Exchange Variable**

The Set Exchange Variable task sets the value of the items within the exchange variable. The **Input** tab of the Set Exchange Variable task is a mapper panel that enables you to set any portion of the exchange variable for the currently selected operation.

See Chapter 6, Transforming Data in a Mediation Exchange, on page 75 for more information about using the mapper panel.

Each Set Exchange Variable task that attempts to configure a portion of the exchange variable resets the value of the entire exchange variable. You can have any number of Set Exchange Variable tasks in the input, output, or fault paths of your mediation flow.

See Set Exchange Variable on page 208 for a description of the tabs on the properties view of this task.
Creating Simple Schemas

You might need a schema for an exchange variable that is not stored in the project. The Create button in the Exchange Schema field opens a simplified schema editor dialog that you can use to create basic schemas. The simplified schema editor creates and stores the XSD file for the schema you create in the specified location in the project.

When creating a complex schema, use the XSD editor in TIBCO Business Studio. See the Eclipse documentation, XSD Developer Guide, for more information about the XSD editor within TIBCO Business Studio.
The simplified schema editor has three fields:

- **Schema**—the structure of the schema. Use the buttons to add, move, and delete schema elements. Only elements can be created using this editor. If you must add attributes or create types, use the XSD editor in TIBCO Business Studio. Table 4 describes each of the buttons in the Schema field.

- **Resource Name**—the name of the schema to create.

- **Workspace Location**—the location in the workspace where the schema will be stored. Use the Browse button to locate a folder in another workspace.

**Table 4** Buttons for creating schema elements

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Add Group Button" /></td>
<td>Adds a group to the schema. You can specify a name for the group and the type of group from one of these options:</td>
</tr>
<tr>
<td></td>
<td>• sequence in order—the elements in the group must appear in the order in which they are specified in the schema.</td>
</tr>
<tr>
<td></td>
<td>• choice of one—the group is a choice group where only one of the elements in the group can appear at a time.</td>
</tr>
<tr>
<td></td>
<td>• all in any order—all elements contained in the group can appear in any order.</td>
</tr>
</tbody>
</table>
Table 4  Buttons for creating schema elements

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Button" /></td>
<td>Adds a complex element to the schema that can contain other elements. You can specify a name for the complex element, a type (from another stored schema), and the minimum and maximum number of occurrences of the element.</td>
</tr>
<tr>
<td><img src="image2.png" alt="Button" /></td>
<td>Adds a primitive element to the schema. You can specify a name for the primitive element, a primitive type (string, integer, and so on), the minimum number of occurrences, and the maximum number of occurrences of the element.</td>
</tr>
</tbody>
</table>
| ![Button](image3.png) | Adds a reference element. A reference element refers to a top-level element, allowing elements to be reused by reference. References in otherschema resources are automatically maintained using imports.  

**Note:** The Simplified Schema Editor does not support duplicate namespaces. The assumption is that a given namespace is only imported once, and is associated with a single prefix. |
| ![Button](image4.png) | Moves the currently selected element up one position in the schema. |
| ![Button](image5.png) | Moves the currently selected element down one position in the schema. |
| ![Button](image6.png) | Deletes the currently selected element. |

After creating a schema, click **OK** to save the schema to the project. If a schema with the same name exists, a message asks for permission to overwrite it.
Working with Tasks

Several operations are the same for all tasks within a mediation flow. This section describes the operations that are common to all tasks.

See these chapters for detail on how to work with each task in a mediation flow:

- Chapter 3, Invoking an Operation, on page 49
- Chapter 4, Logging Mediation Exchange Information, on page 53
- Chapter 5, Routing Messages in a Mediation Flow, on page 59
- Chapter 6, Transforming Data in a Mediation Exchange, on page 75
- Chapter 9, Replying to Messages, on page 135
- Chapter 10, Fault Processing in a Mediation Flow, on page 141

See Chapter 12, Reference, on page 175 for more information about any tasks not mentioned in the chapters in the list above.

Most operations described in this section can be undone by using the Edit > Undo menu item.

Adding a Task to a Path

Before you add tasks, expand the palette on the Mediation Flow Editor to show the list of tasks available. See Figure 3 on page 4 for an illustration of the Mediation Flow Editor and the palette of mediation tasks.

Not all tasks can be added to all paths. See the description of each task in Chapter 12, Reference, on page 175 for more information about the type of path where you can use the task.

Single Tasks

To add a task to a mediation path:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Select</td>
<td>Select the task in the palette, and click the path line where you want to add the task.</td>
</tr>
<tr>
<td>Drag</td>
<td>Drag a task from the palette to a path. Hold the mouse button until the cursor is over the path.</td>
</tr>
</tbody>
</table>
When a task can be added to the path, the path line becomes bold and the cursor changes to this icon  to indicate the task can be added.

If the task cannot be added to the path, the cursor changes to this icon  .

If tasks are already on the current path, a vertical line appears on the path to show where the new task will be inserted. Move the cursor before or after the existing task to add the task to a specific location.

**Multiple Tasks**

To add multiple tasks of the same type to the path:

1. Select the task in the palette.
2. Press the Ctrl key while clicking on the path.

Route tasks cause the path to split into sub-paths. If your mediation flow requires routing, add the route tasks to the path first. Adding a route task to a specific location can be difficult when other tasks are already on the path.

**Deleting a Task From a Path**

Deleting a route task deletes all sub-paths and tasks after the route task. See *Chapter 5, Routing Messages in a Mediation Flow, on page 59* for more information about manipulating route tasks.

You can delete Throw Fault and Generate Reply tasks, but the flow becomes untargeted. If you delete these tasks, you must retarget them as necessary.

Catch Fault, Handle Reply, and Send Fault tasks are automatically placed onto fault paths and cannot be deleted.

To delete a task from a path:

<table>
<thead>
<tr>
<th>Option</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mouse</td>
<td>Select the task in the path, right-click while hovering the cursor over the task, and choose <strong>Delete</strong> from the pop-up menu.</td>
</tr>
<tr>
<td>Keyboard</td>
<td>Select the task, and press the <strong>Delete</strong> key on your keyboard.</td>
</tr>
</tbody>
</table>

**Cutting, Copying, and Pasting Tasks**

Some tasks can be copied and pasted onto the same or other paths.

To copy or cut and then paste a task:
1. To copy the task, select the task in the mediation flow and either press Ctrl+C, or right-click over the task and select Copy from the pop-up menu.

   To cut the task from the path instead of copying it, either press Ctrl+X, or right-click over the task and select Cut from the pop-up menu.

2. Select the path where you want to paste the task. The path becomes bold.

3. Either press Ctrl+V, or right-click while hovering the cursor over the path and select Paste from the pop-up menu.
Mediation Components

Composite Development provides a complete description of composites and components and how they operate within the TIBCO ActiveMatrix architecture. You should be familiar with the procedures in that manual before attempting to work with mediation components. This section describes the specific configuration options for mediation components within a ActiveMatrix composite.

Figure 9  Relationship between a mediation flow and elements of a composite

Mediation flows provide implementations for mediation components:

- Each mediation interface becomes a component service of the mediation component.
• Each target interface becomes a component reference of the mediation component.

• Each mediation flow property becomes a component property. You can override the values specified for mediation properties at the component or composite level.

You can create wires between composite services and component services, to provide bindings for mediation service consumers. You can also create wires between component references and composite references, to provide bindings to actual service providers for target interfaces. You can also create wires from other component services and references to and from the component services and references for a mediation component.

### Working with Mediation Components

When you develop your application, you can begin by creating mediation flows, or you can begin by creating mediation components. You can easily add a mediation flow to a new mediation component, or you can easily generate a mediation flow from a mediation component.

<table>
<thead>
<tr>
<th>Starting Point</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add a mediation flow to a mediation component</td>
<td>1. Specify the mediation flow on the <strong>Implementation</strong> tab of the component.</td>
</tr>
<tr>
<td></td>
<td>2. Use the <strong>Browse</strong> button to locate the correct mediation flow.</td>
</tr>
<tr>
<td>Create a new mediation flow from an existing component</td>
<td>Perform one of these actions:</td>
</tr>
<tr>
<td></td>
<td>• Right-click the component and choose <strong>Quick Fixes &gt; Generate Mediation Implementation</strong>.</td>
</tr>
<tr>
<td></td>
<td>A new mediation flow is created in the Mediation Flows special folder for the component.</td>
</tr>
<tr>
<td></td>
<td>• Add either a component service or a component reference to the component. A dialog opens, asking you to provide a name for a new mediation flow.</td>
</tr>
</tbody>
</table>

TIBCO ActiveMatrix Service Grid Mediation Component Development
**Starting Point**
Create a new composite from an existing mediation flow

**Procedure**

1. Right-click the mediation flow in the Project Explorer.
2. Choose **Mediation Flow > Generate Composite** from the menu.
   
   A composite is created with the same name as the mediation flow.

---

**Intents and Policies**

Intents set for the mediation are propagated to the component reference. A default policy set is also created. You can edit this policy set to specify parameters of the policy. See *Composite Development* for more information.

---

**Mediation Component Properties View**

Mediation adds the **Implementation** tab to the Properties view of mediation components.
A component’s **Implementation** tab provides the **Mediation Flow** field. Use the **Browse** button to select the mediation flow in the project that provides the implementation for this component.

For information on working with component properties, see *Composite Development*. 
Chapter 3 Invoking an Operation

This chapter describes using the Invoke Operation task to call or invoke another operation and receive a reply within a mediation path. See Invoke Operation on page 182 for reference information about this task.

Topics

- Overview of the Invoke Operation Task, page 50
- Configuring Invoke Operation tasks, page 51
Overview of the Invoke Operation Task

The Invoke Operation task enables you to call an operation of any interface during processing of an input, output, or fault path. The Invoke activity can choose any operation from any interface in the target interface list.

For example, you can invoke an operation on the input mediation path and use the data in the reply message in subsequent tasks in the input path before the mediation flow invokes the specified target operation.

These examples describe use cases for the Invoke Operation task:

- Invoking a service to retrieve information, such as item price for a purchase order, a zip code for a city, or a shipping quote from a shipping service.
- Coordinating with non-automated processes, such as invoking a service to send an email message after the target operation returns a reply message.
- Basic orchestration with other services, such as invoking an approval service before invoking a target operation to allow a merchandise return.

An invoked operation can be either a one-way or request-response message exchange pattern.

If the invoked operation uses the request-response pattern, the mediation flow suspends execution until a reply is received from the invoked operation. The reply message from an invoked operation is placed in the mediation exchange in an element corresponding to the name of the Invoke Operation task. Subsequent tasks in the path can then access the reply message.
Configuring Invoke Operation tasks

The operation to be invoked by the Invoke Operation task must be contained in an interface in the target interface list.

To configure an Invoke Operation task:

1. Add an Invoke task to a path.
2. Select Properties view > General tab > Target Operations, and open the drop-down list to select an operation to invoke.

   You can also press the Shift key and drag the task onto the operation in one of the target interfaces. (Dragging a task without using the Shift key rearranges the task on a path.)

When the Target Operation is selected for the Invoke Operation task, a green hint line appears to indicate which operation the task invokes.

Input Tab

The Invoke Operation task requires an input message for the invoked operation. To construct the input message, use the Input tab of the Invoke tab. The Input tab is a mapper panel, similar to the mapper available in the Transform task. See Chapter 6, Transforming Data in a Mediation Exchange, on page 75 for more information about using the mapper panel.

Figure 10   The Input tab of an Invoke Operation task
Output Tab

When the message exchange pattern of the invoked operation is request-reply, the Output tab displays a static schema tree to represent the output message of the invoked operation.

If the message exchange pattern of the invoked operation is one-way, the Output tab of the Invoke Operation Mediation Task displays No Output Configured. The Output tab of the Mediation Operation is empty.

Fault Handling for Invoke Operations

Faults declared by an operation that an invoke activity references are caught and processed on the fault path. See Overview of Faults on page 142 for more information.

Invoking Operations on Dynamic Interfaces

You can use the Invoke Operation task to invoke operations contained in a dynamic target interface. Dynamic target interfaces require a Set Dynamic Reference task that specifies the actual service to invoke. See Chapter 8, Routing Requests Dynamically at Runtime, on page 121 for more information about dynamic target interfaces.

If the Invoke Operation receives a fault from the target service, the fault flow of the mediation is activated. The Catch Fault task has all target faults that might be generated by all of the invoke tasks, so you can mediate faults that are returned.
Chapter 4  

Logging Mediation Exchange Information

This chapter describes using the Log task to send information from the mediation exchange to the log appender.

See Log on page 184 for reference information about the Log task. See Mediation Exchange on page 10 for more information about the mediation exchange.

Topics

- Overview of Log Tasks, page 54
- Configuring a Log Task, page 55
- Configuring Mediation Appenders and Loggers, page 58
Overview of Log Tasks

Log tasks allow mediation flows to send data to a file (appender). By default, the appender for the mediation task is not configured. See Configuring Mediation Appenders and Loggers on page 58 for information about configuring the appender for the Log mediation task.

You can place a log task on any input, output, or fault path. You can configure the log task to send any or all of these items to the log file:

- **Mediation flow properties** are the properties defined for the mediation flow. These properties are defined either on the Properties tab of the mediation flow, or in the composite or component containing the mediation flow. You can select all mediation flow properties, or you can select individual properties to log.

- **Mediation flow context** logs message context such as component name and mediation flow information, if the Mediation Flow Context option is set on the Advanced section of the mediation operation’s General tab.

- **Message context** is information about the transport or security details of the message. See Working with Message Context Properties on page 30 for information about the Mediation Flow Context option.

- **Message data** is the content of the message. Some tasks, such as custom tasks or Transform tasks, can change the content of the message. The Log task can be used to output the message content at any point in the mediation flow. You can use this information for debugging, auditing, or other purposes.

- **Contributed data** Mediation tasks, such as the Transform task or custom tasks, can add or contribute data to the mediation exchange. For example, the Log task can be used to output any data that previously executed mediation tasks contributed to the mediation exchange.

- **Exchange Variable** If the mediation operation has an exchange variable set, the exchange variable appears as one of the items to log. If Log All Items is selected, the Exchange Variable is automatically logged if it is used.
Configuring a Log Task

After you add a log task to a path within a mediation flow, you specify the type of information you want to log. The Log tab of the Log task configures the information to send to the log file.

To configure a log task, set options on the General tab:

1. Give the task a name and description.
2. Choose the role level of logging for the task.
3. Set the Use Transform Data option.

   This option controls the appearance of the Log tab, where you configure the information to send to the log file.

   If you do not select the Use Transform Data option, the standard Log tab appears. See Using the Standard Log Tab on page 55. The Use Transform Data option is cleared by default.

   If you select the Use Transform Data option, the mapper Log tab appears. See Using the Mapper Log Tab on page 56.

For reference information about the Log task settings, see Log on page 184.

Using the Standard Log Tab

If Use Transform Data is not set (the default), the Log tab shows top-level message information, and you select the items to send to the log file.

To send all available information to the log file, select the Log All Items box.
Using the Mapper Log Tab

If the **Use Transform Data** option is set on the **General** tab of the Log task, the **Log** tab appears as a mapping panel. You can use this panel to build custom log messages.

This example shows part of a message mapped to the `message` element:

The `messageID` element is useful if you need to specify a code, or map from a code that is included in a message. See **Mapping Information for Custom Log**
Messages on page 186 for detailed information about the messageID element.

You can set a property on the mediation flow for the Log task to use at run-time, to override the Log Role setting in the General tab if they are different. For example, you might set the Log Role to debug during development, but set the run-time property to info.

See Chapter 6, Transforming Data in a Mediation Exchange, on page 75 for more information about using a mapping panel.
Configuring Mediation Appenders and Loggers

By default, the runtime informational (INFO), warning (WARN) and error (ERROR) messages logged by the mediation component or the mediation log tasks are sent to the log file of the ActiveMatrix node or the associated appender.

Using ActiveMatrix Administrator to specify a unique appender for the mediation component or the mediation log tasks is also possible. The Administrator allows the user to configure logger and corresponding appenders at application or component level. (Refer to Administration for more details on application or component logger configuration).

Two mediation loggers are available:

- `com.tibco.amx.it.mediation`
  
  The logger named `com.tibco.amx.it.mediation` is used by the mediation component to log runtime error, warning, informational or debug messages and must be applied to the node were the mediation application is running.

- `com.tibco.amx.it.mediation.logTask`
  
  The logger named `com.tibco.amx.it.mediation.logTask` is used by the mediation log tasks and must be applied to the mediation application. This logger is available only if it has been configured at design time.

To send the data logged by the mediation log tasks to a specific appender, either one of the two logger names can be used in the Administrator to configure the application or component level loggers. However, to isolate the data logged by the mediation log tasks from rest of the mediation component messages, the logger named `com.tibco.amx.it.mediation.logTask` must be used.

By default only the informational (INFO), warning (WARN) and error (ERROR) messages are written to the log file of the node or the associated appender. The Mediation component or mediation log task’s debug (DEBUG) or trace (TRACE) messages are not written to the log file of the node.

To view debug or trace level messages, a logging appender must be configured at a debug level.
Chapter 5  

Routing Messages in a Mediation Flow

This chapter describes how to use route tasks to deliver messages within a mediation flow.

See Route Task on page 197 and XPath Route on page 218 for reference information about the Route tasks.

Topics

- Overview of Route Tasks, page 60
- Routing Cases, page 64
- Variables, page 68
- Routing Conditions, page 70
- Changing Route Tasks to XPath Route Tasks, page 73
Overview of Route Tasks

Route tasks are used to specify that messages can be delivered to different destinations based on values within the message data or within other data in the mediation exchange, such as the security context.

Route tasks enable you to send messages to a specific destination based on conditions that you specify. Data from the mediation exchange, such as the message context or the message body, can be used to construct the routing conditions. For example, you might route incoming messages to a local server from 9:00 a.m. to 5:00 p.m., but outside of those times, route incoming messages to a different server.

Figure 13   An example using a Route task

The example in Figure 13 shows the input path of a system that searches for travel reservations. For the searchHotel mediation operation, incoming messages are routed to the appropriate service, based on the city specified in the search request:

- If the city is that of the requestor, the message is sent to the QueryGDS service.
- If the city is in Asia, the message is sent to the QueryGDS_Asia service.
- If the city is in Europe, the message is sent to the QueryGDS_Europe service.
• If the city is in the United States, the message is sent to the QueryGDS_US service.

• A fault is thrown if the city is not the requestor city, or in Asia, Europe, or the United States.

Figure 14, Output path for a route, and Figure 15, Fault path for a route show the output and fault paths for the example searchHotel mediation operation.

There are two types of route tasks:

• Route task enables you to define basic route conditions.

• XPath Route task allows more flexibility in the expressions you can use to define a route condition.

The type of condition that you must define determines which route task is appropriate for your application. See Routing Conditions on page 70 for more information about the types of conditions that can be used by each route task.

If you create Route tasks and later decide that a more complex routing condition is required, you can easily convert the Route task to an XPath Route task. See Changing Route Tasks to XPath Route Tasks on page 73 for more information.

Paths and Route Tasks

Route tasks can be added only to input paths.

Route tasks send each incoming message to a single destination based on which route case evaluates to true, or to a single destination designated as otherwise if none of the cases evaluate to true.

Paths on the input flow to a target operation correspond to paths on the output and fault flow from that target operation. Paths ending in Throw Fault have a corresponding mediation fault path on the Fault flow. Paths ending in a Generate Reply task have a single, common Handle Reply path on the Output flow.

You can only introduce the route in the input path, and the response (output or fault) always returns to the original requester—the requester that invoked the mediation operation.

Figure 14 shows the output path, and Figure 15 shows the fault path of the route shown in Figure 13.
Mediation tasks can be added to sub-paths after a route activity. Typically, you use a Transform task when the input, output, or fault message schema does not match the mediation operation message schema.
See Chapter 6, Transforming Data in a Mediation Exchange, on page 75 for more information about Transform tasks.

Defining a Route

The steps for defining a route task are the same regardless of the type of route task you are using.

Before creating a route, your mediation operation must contain the mediation interface and one or more target interfaces that contain operations between which you route messages.

To define a route:

1. Add a Route or XPath Route task to an existing input path. If you have not yet created any input paths in your mediation flow, draw the path between your mediation operation and one of the target operations that you want for the destination for the route.

   After a route task is added to an input path, a default case and an Otherwise case are created. Cases are the conditions that are evaluated to determine which sub-path a message takes. The Otherwise case is always present, and is used when all other cases evaluate to false.

2. (Optional) Create more cases for the route, to create sub-paths to other target operations in your mediation flow.

3. Add variables to hold the value of data from the message content or message context. These variables are used in the routing conditions you specify in each routing case.

4. Specify the routing conditions for each case, using the variables that you have defined for the route.

5. Use the Input tab on the route task to map data from the message context or message content to the variables you defined in step 3.

The details of performing each of these steps is discussed in the subsequent sections of this chapter.
Routing Cases

Routing cases define the potential destination for the route. Each case leads to a different potential target operation. You must specify two things for each routing case: a name for the case and the destination to which the case leads.

Each routing case leads to one potential target operation. Target operations cannot be shared among routing cases. The relationship between routing cases and target operations must be one-to-one.

Adding Cases

You can define routing cases in two ways:

<table>
<thead>
<tr>
<th>Starting Point</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input task</td>
<td>1. Click the route task in the input path of the mediation flow.</td>
</tr>
<tr>
<td></td>
<td>2. Drag the cursor to the destination.</td>
</tr>
<tr>
<td></td>
<td>The sub-path is automatically drawn, and a case with a default name is added to the Decision tab.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Decision tab</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1. Open the Decision tab of the route task.</td>
</tr>
<tr>
<td></td>
<td>2. Click the Add Case button on the toolbar.</td>
</tr>
<tr>
<td></td>
<td>The Add Case button creates a case, but it does not lead to a specific target operation. Figure 16 shows the sub-path that is displayed in this situation. The sub-path leads to an error icon.</td>
</tr>
<tr>
<td></td>
<td>See Modifying Case Names or Destinations on page 65 for details on how to change the destination of the routing case to a valid target operation.</td>
</tr>
</tbody>
</table>
Specifying Case Targets in the Decision Table

To specify a target in the route task Decision table:

1. Click inside the cell where the target is located
2. Choose a target from the drop-down list.

Targets you can specify are:
- Target operations that are not already targeted
- Generate Reply or Throw Fault mediation tasks
- End Mediation task for one-way (in-only) operations.
- Route and XPath Route mediation tasks, which enables you to build nested routing structures.

If you retarget a Route task, the entire nested routing structure is replaced.

Modifying Case Names or Destinations

Use the Decision tab on the route task to change the name of the routing case.

To change the routing case name:

1. Click the name in the Case column.
2. Edit the name in the text box.

To change the destination of the routing case:
• Select the option that matches your goal:
  — For selecting a new target operation, use the drop-down list in the **Target Service/Operation** field on the **Decision** tab to specify the new target operation.
  — For newly created routing cases that point to an error icon, click the error icon and drag the cursor to the target operation.
  — For routing cases that point to a valid target operation, click the round ball at the end of the input path, and drag the cursor to the target operation.

### Moving Cases in the List

Cases are evaluated in the order in which they appear in the list. The first case whose condition evaluates to true is taken, so you might need to move cases up or down in the list.

To move a case up the list:
1. Select the row of the case.
2. Click the **Move Up** icon on the toolbar.

To move a case down the list:
1. Select the row of the case.
2. Click the **Move Down** icon on the toolbar.

### Deleting Cases

You can delete a routing case in either of these ways:

<table>
<thead>
<tr>
<th>Starting Point</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mediation flow</td>
<td>1. Select the sub-path of the routing case in the mediation flow.</td>
</tr>
<tr>
<td></td>
<td>2. Right-click and select <strong>Delete</strong> from the menu, or press the <strong>Delete</strong> key.</td>
</tr>
<tr>
<td>Route task</td>
<td>1. Select the row of the routing case on the <strong>Decision</strong> tab of the route task.</td>
</tr>
<tr>
<td></td>
<td>2. Click the <strong>Delete Case</strong> icon on the toolbar.</td>
</tr>
</tbody>
</table>
Nesting Multiple Route Tasks

You can use multiple, nested Route tasks to send a message to a target operation. Doing so enables you to create complex mediation paths with multiple conditions.

When you use nested Route tasks, the mediation path shows whether a route goes directly to a target operation, or goes through another Route task first.

When multiple Route tasks are in the mediation path, the task output details (or case paths) from only one of the Route tasks is visible at a time.

This example shows the use of nested Route tasks, with the top-level Route tasks output details visible:

To see the output details of another Route task in the mediation path, click the button next to the Route task icon, or use the outline view to navigate to a specific Route task.

To see the level of a route in a nested set of Route tasks, place the cursor over the button next to the Route task icon.
Variables

Variables hold the data that you use in expressions for each routing case. Variables are managed on the **Decision** tab of the route task.

Adding and Deleting Variables

After a variable is created, you cannot change its name or data type. To change a variable, you must delete the variable and create a new one.

To add or remove a variable:

<table>
<thead>
<tr>
<th>Task</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add a variable</td>
<td>Click the <strong>Add Variable</strong> icon on the toolbar of the <strong>Decision</strong> tab.</td>
</tr>
<tr>
<td></td>
<td>A dialog opens that enables you to specify the variable name and data type. These data types are available:</td>
</tr>
<tr>
<td></td>
<td>• String (default)</td>
</tr>
<tr>
<td></td>
<td>• Integer</td>
</tr>
<tr>
<td></td>
<td>• Boolean</td>
</tr>
<tr>
<td></td>
<td>• Float</td>
</tr>
<tr>
<td></td>
<td>• Double</td>
</tr>
<tr>
<td></td>
<td>• Decimal</td>
</tr>
<tr>
<td>Delete a variable</td>
<td>Click the <strong>Delete Variable</strong> icon on the toolbar.</td>
</tr>
<tr>
<td></td>
<td>A dialog opens that enables you to select the variable you want to delete.</td>
</tr>
</tbody>
</table>

Mapping Data to Variables

After you create a variable, you must map data from the mediation operation to provide a value for the variable. The mapping is performed on the **Input** tab of the route task. This tab is very similar to the **Mapper** tab of a Transform task. For a complete description of how to perform mapping, see Chapter 6, Transforming Data in a Mediation Exchange, on page 75.
In the **Input** tab of a route task, the right-hand panel is labeled **Rule Variables**. The schema in the right-hand panel contains a list of the variables that you have defined for the route task. Use XPath expressions to provide a value for each variable in this schema.

In the example shown in Figure 13 on page 60, one variable named `city` is used to determine the destination of the message. In this example, the city specified in the search request is mapped to the variable named `city`. The value of the variable is then used in routing conditions to determine which target operation should receive the message. Figure 17 shows the mapping for this example.

### Figure 17  Mapping values to variables

The section **Schema Components on page 86** explains that schema components on the left side of the mapper are not validated against the message schema, and their data types are thus not guaranteed. Therefore, data used within XPath expressions on the right side of the mapper is treated as untyped strings. Simple drag-and-drop mappings are not affected, but if you want to perform data type-dependent comparisons or operations, you must use the Constructor Functions on the **Functions** tab (for example, `xsd:int()`) to correctly specify the data type. For example, to add two integers, the XPath expression would be:

```
xsd:int($MessageData/int1) + xsd:int($MessageData/int2)
```
Routing Conditions

Routing conditions determine which sub-path a message takes. Routing conditions are specified in order, and the message is sent along the sub-path corresponding to the first condition that evaluates to true. The Decision tab of the route tab contains the routing conditions for the route.

Routing conditions are XPath expressions, but each type of route task has a different method of specifying routing conditions:

- The Route task enables you to specify basic comparison expressions for each variable you have defined.
- The XPath Route task enables you to use more complex XPath expressions.

The type of route you use depends on the complexity of the routing conditions you need to define.

Conditions for Route Tasks

Route tasks create a simple comparison condition for each variable you have defined. A Route task is useful in situations where a basic comparison of a few variables can be specified.

For example, Figure 18 shows the routing conditions defined for the example shown in Figure 13.

Figure 18  Routing conditions for the Route task

In this example, basic equality comparisons are performed for each case. When the city variable is equal to "Palo Alto", the case named CaseMyCity evaluates to true, and its corresponding sub-path is taken. If none of the routing conditions evaluate to true, the sub-path for the Otherwise case is taken.
Edit Route Task Conditions

To edit a route task condition:

1. Click the cell for each variable and each case.

2. Specify a comparison operator and a constant value for comparison.

   Basic comparison operators are available in a drop-down list in the condition:
   
   — = (equal)
   — != (not equal)
   — < (less than)
   — <= (less than or equal)
   — > (greater than)
   — >= (greater than or equal)
   — =true() (only for variables of type boolean)
   — =false() (only for variables of type boolean)

   All conditions for each case must evaluate to true for the condition to be true.

The Route task is useful in some situations where a basic comparison of a few variables can be specified. Figure 19 shows an example of using more than one variable in routing cases.

Figure 19  Routing with more than one variable

In the example loan application in Figure 19, the operation SimpleLoanPortType/SimpleRequestLoan can be used in two circumstances:

- For loan amounts that are less than or equal to $50,000
- When the applicant has a credit score above 700
If neither of these conditions is true, the LoanPortType/RequestLoan operation that requires more information from the applicant must be used.

Conditions for XPath Route Tasks

XPath Route tasks allow more complex comparisons for each case than does a Route task. You can specify an XPath expression for each case that examines the value of one or more of the variables that you have defined. For example, you can create an expression that specifies a range of time (such as 9AM to 5PM), or you can create an expression that compares two or more variables. Your expressions are not limited to simple comparisons, and you do not need to use any of the variables you have defined in the expressions.

XPath Route tasks are more flexible than Route tasks, but specifying the expression is more complex. You must type the XPath expression in the condition field next to each routing case.

Variables are referenced in the XPath expressions for each routing case by their names. Unlike XPath expressions in the Transform task, you do not need to use a dollar sign to specify the root of the path to the variable. For example, the expression to determine if the city variable is equal to "Palo Alto" would be:

\[ \text{city} = "\text{Palo Alto}" \]

The Transform task has a graphical XPath editor that you can use as a reference for creating XPath functions for the route task. See Using XPath on page 107 and Data/Function Tabs on page 91 for more information about XPath.

As noted in Mapping Data to Variables on page 68, data type-dependent comparisons and operations should use constructor functions to typecast the data.
Changing Route Tasks to XPath Route Tasks

You can change a Route task to an XPath Route task if, for example, you originally create Route tasks then later realize that a more complex routing condition is required. Instead of removing the existing Route task, you can convert a Route task into an XPath Route task.

Converting a Route task to an XPath Route task is a one-way operation. You cannot convert an XPath Route task to a Route task.

To convert a Route task to an XPath Route task:
1. Select the Route task to convert in a mediation flow diagram.
2. Right-click the Route task and select **Convert to XPath Route** from the menu.
   All variables and cases are maintained, and routing conditions are converted to the correct XPath syntax.
3. Change the condition for each case, as necessary.
Chapter 6  Transforming Data in a Mediation Exchange

This chapter describes using Transform tasks to manipulate data in a mediation flow.

Topics

- Overview of Transform Tasks, page 76
- Example of Transformation, page 78
- Contributing Data to the Mediation Exchange, page 82
- Using External Stylesheets for Data Transformation, page 83
- Schema Components, page 86
- Context Panel, page 88
- Message Panel, page 90
- Data/Function Tabs, page 91
- Toolbar, page 97
- Right-Click Menu, page 98
- Mappings, page 104
- Using XPath, page 107
- Transforming XML with Related Tasks, page 110
Overview of Transform Tasks

Transform tasks, used to manipulate data available in a mediation exchange, are necessary when the schema of the input, output, or fault message does not match the schema of the message of the expected recipient.

Transform tasks can enable you to achieve several goals:

- Create a mediation operation that allows new clients to use legacy services with different schemas. Your new client might need a service that returns an integer for salary information, but the legacy service returns a string.

- Contribute data to the mediation exchange for use in subsequent mediation tasks. For example, you might want to place into a string the time a message was sent, the sender of the message, and the value of one of the elements within a message. You can then use a log task to write the contents of that constructed string to the log file.

- Manipulate and store data in the mediation exchange without changing the actual message content.

Transform tasks have an Input tab that contains the expected schema of the recipient’s message and the data available in the mediation flow.

Figure 20 The Input tab of a Transform task

The message panel contains an XSLT stylesheet that creates the message that the recipient expects. The message panel initially displays the expected schema of the recipient’s message, to give you hints about constructing the message.
The Mediation Operation Context panel contains the data available from the message sender. You can drag items from the context panel to the message panel to perform simple mappings. More complex mappings are also possible through the **XPath expression** field and by using the right-click menu in the message panel to add XSLT statements.

XPath and XSLT are standard tools for data transformation. Extensive knowledge of XPath or XSLT is not necessary to use the mapper effectively. You can accomplish most transformation usage cases with information available in this chapter and in the Help text available for each XPath function in the product.

If you perform more complex transformations, however, it is helpful to have some detailed references on XPath and XSLT. It is beyond the scope of this manual to provide a complete reference for these tools. You can find the complete XPath and XSLT specification at [www.w3.org](http://www.w3.org), and several third-party commercial books are available on both XPath and XSLT.
Example of Transformation

In this example of using the Transform task, the mediation flow is for a travel reservation service. The mediation operation exposes the service as a single interface, but the mediation flow routes incoming requests to the appropriate local service based on the location of the hotel. Different continents have different target services that perform the hotel reservation. The schemas for different locations are slightly different, and so some transformation might be necessary.

Figure 21  A travel reservation mediation flow: the input path

In this example, Transform tasks are required when requests come in for any city other than the local city, because the schemas for the other target operations are different from the mediation operation. Figure 22 shows the transform task for the case when a reservation is requested for an Asian city.
Figure 22 shows that the schemas for the mediation operation and the target operation are similar, except that the mediation operation has an element named `nearAddress` and the corresponding element in the target operation is named `landmark`. For all other elements, you can drag and drop the data component in the mediation operation to the corresponding data component in the target operation, and the appropriate XPath expression is placed in the XPath field.

For the `nearAddress` and `landmark` elements, you might need to manipulate the data to transform it to the expected format.

In this example, the `nearAddress` element contains the name of the location separated by a comma, followed by the actual address. The `landmark` element is expecting only the name of the location. To make the data match the target operation’s expectations, you need to take the substring of the `nearAddress` element up to the comma that separates the name of the location from the address.

To accomplish this goal:

1. Click the **XPath Expression** field next to the `landmark` element in the target operation message schema.
   
   The field expands to a larger text box so that you can edit the expression easily.

2. Click the **Functions** tab on the at the top of the context panel.

3. Expand the String functions folder in the functions list and locate the `substring-before` function.

4. Drag the `substring-before` function into the XPath Expression window.

Figure 23 shows the results of adding the `substring-before` function to the XPath expression.
When you drag functions from the **Functions** tab to the XPath Expression window, the function shows markers in double angle brackets (for example, `<<string>>`) for completing the function. You can drag data components and constants from the **Data Source** or **Constants** tab to complete the function. You can also type in the XPath Expression window to replace the markers manually.

In this example, you would replace the `<<before-string>>` marker with a comma and then drag the `nearAddress` element onto the `<<string>>` marker. **Figure 24** shows dragging the data components into an XPath function.
Figure 25 shows the completed XPath function for this example.

Figure 25  The completed transformation example

<table>
<thead>
<tr>
<th>Target/Operation/Message</th>
<th>XPath Expression</th>
</tr>
</thead>
<tbody>
<tr>
<td>MessageData</td>
<td></td>
</tr>
<tr>
<td>searchHotelRequest</td>
<td></td>
</tr>
<tr>
<td>parameters</td>
<td></td>
</tr>
<tr>
<td>country</td>
<td>$MessageData/nosearchHotelRequest/parameters/nosearchHotel/country</td>
</tr>
<tr>
<td>city</td>
<td>$MessageData/nosearchHotelRequest/parameters/nosearchHotel/city</td>
</tr>
<tr>
<td>state</td>
<td>$MessageData/nosearchHotelRequest/parameters/nosearchHotel/state</td>
</tr>
<tr>
<td>landmark</td>
<td>substring before($MessageData/nosearchHotelRequest/parameters/nosearchHotel/nearAddress, &quot;,&quot;)</td>
</tr>
<tr>
<td>checkInDate</td>
<td>$MessageData/nosearchHotelRequest/parameters/nosearchHotel/checkInDate</td>
</tr>
<tr>
<td>checkOutDate</td>
<td>$MessageData/nosearchHotelRequest/parameters/nosearchHotel/checkOutDate</td>
</tr>
<tr>
<td>numberOfRooms</td>
<td>$MessageData/nosearchHotelRequest/parameters/nosearchHotel/numberOfRooms</td>
</tr>
<tr>
<td>nonSmoking</td>
<td>$MessageData/nosearchHotelRequest/parameters/nosearchHotel/nonSmoking</td>
</tr>
</tbody>
</table>

More complex transformations are possible with the features available in the **Input** tab. The following sections in this chapter describe the different parts of the **Input** tab in further detail.
Contributing Data to the Mediation Exchange

The Transform task can perform two functions:

- Modify the message data within the mediation exchange
- Contribute new data to the mediation exchange.

On the General tab of the Transform task, the checkbox labeled **Contribute Output to Mediation Exchange** specifies how the Transform task results are handled:

- If **Contribute Output to Mediation Exchange** is cleared, the results of the transformation is used to construct a new message. This option is cleared by default.
- If **Contribute Output to Mediation Exchange** is selected, the results of the Transform task are added to the mediation exchange as a new data element. The new data element is available to subsequent mediation tasks along the same path, and the name of the data element is the same as the name assigned to the Transform task.

The **Contribute Output to Mediation Exchange** option is automatically selected if you use an external stylesheet for data transformation. See Using External Stylesheets for Data Transformation on page 83.
Using External Stylesheets for Data Transformation

You can use an external, third-party XSLT stylesheet for data transformation using the Transform task. This enables you specify the transformation mapping in your workspace, outside the mediation flow.

You can specify an external XSLT stylesheet for transformation in two ways, using reference types:

- A static reference enables you to select a single (static) stylesheet from a folder in your project.
- A dynamic reference enables you to select a set of stylesheets from a folder in your project. At run-time one of the stylesheets in the list is used dynamically, based on the value provided for the stylesheetURI element in the Input tab of the mediation task.

For example, if the folder specified for the dynamic reference is MySOAPProject/Service Descriptors and the stylesheet is in the folder MySOAPProject/Service Descriptors/folder1/sample.xsl, the value that must be provided for the stylesheetURI element would be folder1/sample.xsl.

When you specify a folder for dynamic reference, ActiveMatrix recursively includes the stylesheets under this folder and its sub-folders.

The stylesheet for a reference must be located in the same project as the mediation flow that uses it.

Specifying an External Stylesheet for Data Transformation

1. On the General tab of the Transform task, select the checkbox labeled Use External Stylesheet.

The stylesheet selection options open on the General tab:

![Screenshot of Transform Mediation Task window]

TIBCO ActiveMatrix Service Grid Mediation Component Development
2. Open the **Input Style** drop-down menu and specify how the XML will appear:

   - **Text** Specified with a string.
   - **Binary** Specified with a binary value.
   - **Tree** Specified with a type of *any*, enabling you to transform data that is already in an XML document.

**Contribute Output to Mediation Exchange** is automatically selected for this type of transformation, which prevents the MessageData from being overwritten when an external transformation is used. Also, the input and output of the transformation task always match the Input Style you select. For example, if the input is text, the output is also be text.

3. Open the **Stylesheet Reference Type** drop-down menu of the **General** tab and select the type of reference for the Transform task to use:

   **Static reference type**

   a. Browse and select a stylesheet from the stylesheets you have saved in your project.

   b. Open the **Input** tab of the Transform task and map the data, so that when the data arrives, the value is transformed using the specified stylesheet:
Dynamic reference type:

a. Open the **Dynamic Stylesheet Folder** drop-down menu of the **General** tab and select the folder where one or more stylesheets are located.

b. Open the **Input** tab of the Transform task and provide the stylesheet name as a parameter in the message data:

![Image of Transform Task]

At run-time, ActiveMatrix searches for this name in the folder you specified.

If the xsl file is in a sub-folder, the name must include the relative path name. For example, in the case where the xsl file is located in `company/dept/app.xsl`, the top-level folder (in this example, `/toplevel`) is prepended to locate the exact location for the file in the project:

`/toplevel/company/dept/app.xsl`

See **Transform on page 211** for reference information about the Transform task.
### Schema Components

The message panel and context panel each contain schemas that contain data components. Table 5 describes the icons used in these schemas.

The icons represent the general data type of the component. To see the exact data type, hover the cursor over a component to open a pop-up.

**Table 5  Icons for schema components**

<table>
<thead>
<tr>
<th>Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="complex.png" alt="Complex Element" /></td>
<td>Complex element that is a container for other datatypes. This is also called a parent in the schema tree.</td>
</tr>
<tr>
<td><img src="string.png" alt="String or Character Value" /></td>
<td>String or character value.</td>
</tr>
<tr>
<td><img src="integer.png" alt="Integer Value" /></td>
<td>Integer value.</td>
</tr>
<tr>
<td><img src="decimal.png" alt="Decimal Value" /></td>
<td>Decimal (floating point) value.</td>
</tr>
<tr>
<td><img src="boolean.png" alt="Boolean Value" /></td>
<td>Boolean value.</td>
</tr>
<tr>
<td><img src="date.png" alt="Date Value" /></td>
<td>Date value.</td>
</tr>
<tr>
<td><img src="time.png" alt="Time Value" /></td>
<td>Time value.</td>
</tr>
<tr>
<td><img src="binary.png" alt="Binary Value" /></td>
<td>Binary (base 64()) value.</td>
</tr>
<tr>
<td><img src="choice.png" alt="Choice of Multiple Values" /></td>
<td>Choice of multiple values. The actual data value can be one of a specified set of datatypes.</td>
</tr>
</tbody>
</table>

To improve performance, data contained within schema components in the left side of the mapper are not validated against the message schema for the operation. Therefore, data used within XPath expressions on the right side of the mapper are treated as untyped strings.

To perform datatype-dependent comparisons or operations, use the Constructor Functions on the **Functions** tab (for example, `xsd:int()`) to correctly specify the datatype. For example, to add two integers, the XPath expression would be: `xsd:int($MessageData/int1) + xsd:int($MessageData/int2)`
Qualifier Characters

Schema data components can have additional characters to the right of the element name that specify additional information. If there is no qualifier, the schema component is required and you must provide a mapping that results in a value for the schema component.

Table 6  Qualifier characters for schema components

<table>
<thead>
<tr>
<th>Qualifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>Element is required.</td>
</tr>
<tr>
<td>?</td>
<td>Element is optional.</td>
</tr>
<tr>
<td>*</td>
<td>Element repeats zero or more times.</td>
</tr>
<tr>
<td>+</td>
<td>Element repeats one or more times.</td>
</tr>
</tbody>
</table>
Context Panel

The name of the context panel is based on the type of path where the Transform task appears:

<table>
<thead>
<tr>
<th>Type of Path</th>
<th>Name of Context Panel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Mediation Operation Context</td>
</tr>
<tr>
<td>Output</td>
<td>Target Operation Context</td>
</tr>
<tr>
<td>Fault</td>
<td>Mediation Fault Context</td>
</tr>
</tbody>
</table>

The context panel always displays the schemas that define the data for the current mediation properties, message flow context, and message data. Regardless of the type of path, the schema of the mediation properties and message flow context are always the same. The schema for the message data varies depending upon the schema of the recipient’s expected message.

Table 7  Schema for message properties and message flow context

<table>
<thead>
<tr>
<th>Schema Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MediationFlowProperties</td>
<td>This schema component contains an element named properties that is of type complex that contains the properties defined on the Properties tab of the mediation flow. See Mediation Flow Properties on page 22 for more information.</td>
</tr>
<tr>
<td>MessageFlowContext</td>
<td>This schema component contains the defined context parameters. See Working with Message Context Properties on page 30 for more information.</td>
</tr>
</tbody>
</table>
Table 7  Schema for message properties and message flow context

<table>
<thead>
<tr>
<th>Schema Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MessageData</td>
<td>The MessageData component contains the message of the expected recipient. For example, for input paths this component contains the schema of the input message of the mediation or the target operation. For output paths, this component contains the schema of the reply message of the mediation or the target operation. Similarly, for fault paths this component contains the schema of the fault message. For fault paths, this component contains a choice element that contains either one of the faults returned by the target operation or a generic Undeclared fault message.</td>
</tr>
</tbody>
</table>
Message Panel

The message panel contains the schema of the message that the recipient expects. The name of the message panel is based on the type of path where the Transform task appears:

<table>
<thead>
<tr>
<th>Type of Path</th>
<th>Name of Message Panel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Target Operation Context</td>
</tr>
<tr>
<td>Output</td>
<td>Mediation Operation Context</td>
</tr>
<tr>
<td>Fault</td>
<td>Mediation Fault Context</td>
</tr>
</tbody>
</table>

You can use the data in the schemas from the context panel to construct the content of the message expected by the receiver. The message panel is actually an Extensible Stylesheet Language Transformation (XSLT) template that specifies how data will be transformed to produce the expected message.

You do not need detailed knowledge of XSLT to create the mappings for the message. Most mappings can be accomplished by simple dragging from the context panel to the message panel, and also possibly using a few XPath functions for simple data manipulation. If you want to see the XSLT template that is created from your mappings, click the Show Edit Tab icon on the toolbar, then click the XSLT Source tab at the top of the XPath editor dialog.

See Data/Function Tabs on page 91 and Mappings on page 104 for more information about using XPath functions and creating mappings.
**Data/Function Tabs**

Use the tabs at the top of the context panel to select items to drag to the message panel.

**Table 8  Data/Function tabs**

<table>
<thead>
<tr>
<th>Tab</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data Source</td>
<td>Contains the schemas for the mediation flow properties, message flow context, and message data. This tab is selected by default when you view the Input tab. See Context Panel on page 88 for more information.</td>
</tr>
<tr>
<td>Functions</td>
<td>Contains a set of XPath functions organized into related functional groups. XPath (XML Path Language) is an expression language developed by the World Wide Web Consortium (W3C). XPath functions perform data manipulation, such as mathematical functions, string manipulation, or logic operators. You can select and drag XPath functions in this tab to the XPath expression field or to the Show Edit Tab dialog in the message panel. Each function has help text to describe the function’s use and syntax. The help for the function is displayed below the function list in the Functions tab. Table 9, TIBCO XPath Functions describes the functions added by Mediation.</td>
</tr>
<tr>
<td>Constants</td>
<td>Contains constants such as whitespace or symbol characters that can be used in XPath expressions.</td>
</tr>
</tbody>
</table>

When you drag data, a function, or a constant to the right-hand panel and hover over an existing expression in an XPath editing window, the background color of the text underneath the cursor changes. The new color indicates the result of placing the item at that point:

- Light turquoise - The highlighted text is the first parameter of the dropped function.
As noted in Schema Components on page 86, data in schema components on the Data Source tab are not validated and checked against the types in the message schema. Therefore, data is coerced into an untyped string. The Constructor Functions on the Functions tab must be used on data to correctly evaluate most functions and operators.

### Table 9  TIBCO XPath Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>base64ToString</td>
<td>Converts a base64 binary encoded string to a string using the specified encoding. If encoding is not specified, UTF-8 is used.</td>
</tr>
<tr>
<td></td>
<td><strong>Template</strong></td>
</tr>
<tr>
<td></td>
<td>base64ToString(&lt;&lt; encodedString &gt;&gt;, &lt;&lt; optional encoding &gt;&gt;)</td>
</tr>
<tr>
<td></td>
<td><strong>Return Type</strong></td>
</tr>
<tr>
<td></td>
<td>string</td>
</tr>
<tr>
<td>stringToBase64</td>
<td>Converts a string to a base64 binary encoded string.</td>
</tr>
<tr>
<td></td>
<td><strong>Template</strong></td>
</tr>
<tr>
<td></td>
<td>stringToBase64(&lt;&lt; stringToEncode &gt;&gt;)</td>
</tr>
<tr>
<td></td>
<td><strong>Return Type</strong></td>
</tr>
<tr>
<td></td>
<td>string</td>
</tr>
<tr>
<td>hexToString</td>
<td>Converts a hex string to a string using the specified encoding. If encoding is not specified, UTF-8 is used.</td>
</tr>
<tr>
<td></td>
<td><strong>Template</strong></td>
</tr>
<tr>
<td></td>
<td>hexToString(&lt;&lt; encodedString &gt;&gt;, &lt;&lt; optional encoding &gt;&gt;)</td>
</tr>
<tr>
<td></td>
<td><strong>Return Type</strong></td>
</tr>
<tr>
<td></td>
<td>string</td>
</tr>
</tbody>
</table>
Creating Custom XPath Functions

This section describes how to create and deploy custom XPath functions.

To create the plug-in for the custom XPath function:

1. Run TIBCO Business Studio from the Start menu. For example, select Start > All Programs > TIBCO_HOME > TIBCO Business Studio N.N > Studio for Designers.

2. Select File > New > Project....

3. In the New Project dialog under Plug-in Development, select Plug-in Project and click Next.

4. Specify a name for the project that reflects the XPath functions (for example, My Custom Functions).

   Accept all other defaults and click Next.

5. On the Plug-in Content page, locate the Plug-in Options group, and clear these options:

   — Generate an activator, a Java class that controls the life cycle of the plug-in.

   — This plug-in will make contributions to the UI

6. Accept all other defaults, and click Next.

7. In the Templates page, select Custom XPath Function Wizard and click Next.

Table 9  TIBCO XPath Functions

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>stringToHex</td>
<td>Converts a string to a hex encoded string.</td>
</tr>
</tbody>
</table>

**Template**

stringToHex(<< stringToEncode >>)

**Return Type**

string

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>timestamp</td>
<td>Returns the number of milliseconds since midnight, January 1, 1970 UTC, at the instance of the call to this function</td>
</tr>
</tbody>
</table>

**Template**

timestamp()

**Return Type**

long
8. In the Custom XPath Function Group dialog box, provide values:
   — Category: This is the name of the category that will include the custom
     XPath function.
   — Prefix: The prefix for the functions
   — Namespace: The namespace for the functions.
   — Help Text: The description of the functions.
9. Click Next to continue.
   The XPath Function Group Creation Section dialog is displayed. Here you
   specify the function and function parameters.
10. Click the Add button located on the right side of the XPath Function table.
    Provide values for the function:
    — Name: The name of the function.
    — Return Type: The return type of the function.
    — Description: The description of the function.
11. Click the Add button located on the right side of the XPath Function
    Parameters table.
    Specify values to define the parameter:
    — Name: The name of the parameter.
    — Type: The data type of the parameter.
    — Optional: Select the check box if the parameter is optional.
12. Click Finish.
13. TIBCO Business Studio opens the Open Associated Perspective dialog, which
    asks if you want to open the Plug-in Development perspective.
    Optionally, select the check box Remember my decision. Click Yes. TIBCO
    Business Studio opens the custom XPath function plug-in and the Plug-in
    Development perspective.

Along with the custom XPath plug-in, a SOA Project <plug-in project
name>.deploy.soa is created.

Your custom code is written in <plug-in project name>
\<plug-in project name>\src\<category name>.java.

**Exporting Custom XPath Functions**

To install a custom XPath function in TIBCO Business Studio:
1. Create a feature project.

   Specify the plug-in to package into the new feature.


2. Export the feature project.

   Make sure you select the check box for the Generate metadata repository option.


3. Install the feature using Help > Install New Software.

   Specify the location where you exported the feature project. Unselect the check box for the Group items by category option which will then list the feature project.

   The custom XPath function is ready for use and can be accessed from the Input path of the data transform function.

![Custom XPath Function](image-url)
Deploying Custom XPath Functions

After the file `<plug-in project name>\src\<plug-in project name>\<category name>.java` is updated with the custom code, the deployable artifacts can be generated.

To create deployable artifacts:

1. Make sure the Target Platform points to TIBCO ActiveMatrix Runtime and no errors occur in the custom XPath function plug-ins.

   See Composite Development for information on switching the Target Platform.

2. In the Project Explorer pane, expand the `<plug-in project name>.deploy.soa` project.

3. Expand the Composites folder.

4. Right-click `<plug-in project name>.apt.composite` and click Create DAA.

The Create Deployment Archive wizard is invoked. Refer to Composite Development for more information on using this wizard.

Deploy this deployment archive, the DAA, like any other SOA project. Refer to Administration for information on uploading and deploying the DAA.

Testing Custom XPath Functions

To test a custom XPath function in RAD by creating a Run As/Debug As configuration:

1. Add one of the following to the Functions list along with the main composite:

   — A composite generated by the Custom XPath Function wizards to the list
   — A DAA created from the composite

   Make sure that the composite or DAA that holds Custom XPath Function is at the top of the list of Composite/DAA(s), before the SOA DAA/Composite.

2. Select Apply and Run/Debug.
The toolbar contains icons to perform various functions in the mapper.

Table 10  Mapper toolbar buttons

<table>
<thead>
<tr>
<th>Button</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Icon" /></td>
<td>Pins the property view to the current selection.</td>
</tr>
<tr>
<td><img src="image2" alt="Icon" /></td>
<td>Click this button to view errors for the selected element or children.</td>
</tr>
<tr>
<td><img src="image3" alt="Icon" /></td>
<td>Click to remove the selected mapping. This button is available only when a mapping is selected in the message panel. If you remove the mapping when a parent node in the schema tree is selected, all mappings for child nodes of the parent are also removed.</td>
</tr>
<tr>
<td><img src="image4" alt="Icon" /></td>
<td>Deletes XSLT statements that you have added using the right-click menu, such as variables, comments, or choose statements. This button is available only when a statement you have added is selected.</td>
</tr>
<tr>
<td><img src="image5" alt="Icon" /></td>
<td>Opens the Show Check and Repairs dialog. See Repairing Incorrect Mappings on page 105 for more information.</td>
</tr>
<tr>
<td><img src="image6" alt="Icon" /></td>
<td>Opens a larger Show Edit Tab XPath editing window for the selected element in the message panel. The window gives you access to a larger XPath viewer, the XSLT source, and controls that enable you to further edit the XSLT statements. Click this icon a second time to make the Show Edit Tab XPath editing window disappear.</td>
</tr>
</tbody>
</table>
Right-Click Menu

Right-clicking on a data component in the message panel opens a popup menu with several choices.

Table 11  Right-click menu in the message panel

<table>
<thead>
<tr>
<th>Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Show Mappings</td>
<td>Expands the selected component to show all sub-components with mappings. Also expands any data components in the left-hand panel that correspond to mappings so that all mapping lines are shown. If no component is selected, the operation is performed on the root of the schema tree.</td>
</tr>
<tr>
<td>Show Errors</td>
<td>Expands the selected data component to show all sub-components that have errors. If no component is selected, the operation is performed on the root of the schema tree.</td>
</tr>
<tr>
<td>Expand All</td>
<td>Expands all sub-components of the selected data component. If no component is selected, the operation is performed on the root of the schema tree.</td>
</tr>
<tr>
<td>Surround With &gt; Choose</td>
<td>Surrounds the selected data component with a Choose statement. See Choose Statements on page 99 for more information.</td>
</tr>
<tr>
<td>Surround With &gt; Surround with If</td>
<td>Surrounds the selected data component with an If statement. See If Statements on page 100 for more information.</td>
</tr>
<tr>
<td>Surround With &gt; Surround with ForEach</td>
<td>Surrounds the selected data component with a For Each statement. See For Each Statements on page 101 for more information.</td>
</tr>
<tr>
<td>Surround With &gt; Surround with ForEach Group</td>
<td></td>
</tr>
</tbody>
</table>
### Choose Statements

Choose statements enable you to conditionally specify the mapping based on an expression. Choose statements consist of a When clause to specify the condition you want to test, the mapping you want to perform if the condition is true, and an Otherwise clause to contain a mapping to perform if no conditions evaluate to true.

To surround a component with a Choose statement:

1. Select the component to surround, right-click, and choose **Surround > Surround with Choose...** from the menu.

2. In the Surround With Choose dialog, enter the number of When conditions to test against, and also specify whether to include an Otherwise clause for any unhandled conditions.

3. For each When clause, create an XPath expression that evaluates to a boolean.

<table>
<thead>
<tr>
<th>Menu Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Child &gt;</td>
<td>Adds a sub-component to the selected data component. The child component will be a variable. Variables can be set to a constant value and used in other mappings in the message panel. See <strong>Variables on page 102</strong> for more information.</td>
</tr>
<tr>
<td>Variable</td>
<td></td>
</tr>
<tr>
<td>Add Child &gt;</td>
<td>Adds a sub-component to the selected data component. The child component will be a comment. See <strong>Comments on page 103</strong> for more information.</td>
</tr>
<tr>
<td>Comment</td>
<td></td>
</tr>
<tr>
<td>Add Sibling &gt;</td>
<td>Adds a data component at the same level as the selected component. The new component will be a variable. Variables can be set to a constant value and used in other mappings in the message panel. See <strong>Variables on page 102</strong> for more information.</td>
</tr>
<tr>
<td>Variable</td>
<td></td>
</tr>
<tr>
<td>Add Sibling &gt;</td>
<td>Adds a data component at the same level as the selected component. The new component will be a comment. See <strong>Comments on page 103</strong> for more information.</td>
</tr>
<tr>
<td>Comment</td>
<td></td>
</tr>
<tr>
<td>Toolbar icons</td>
<td>The selections from the toolbar are also available in the right-click menu. See <strong>Toolbar on page 97</strong> for more information about these options.</td>
</tr>
</tbody>
</table>

### Table 11  Right-click menu in the message panel

<table>
<thead>
<tr>
<th>Menu Item</th>
<th>Description</th>
</tr>
</thead>
</table>
4. Under each When clause, provide the XPath expression for the mapping that occurs if the When condition evaluates to true.

5. If an Otherwise clause is specified, provide an XPath expression for the mapping that occurs if no When conditions evaluate to true.

An example of using a Choose statement is when more than one fault message is handled by the same Catch Fault task. Figure 26 shows a Transform task on a fault path that handles two faults. The Choose statement specifies that when the searchHotel_faultMsg is returned, send the value of the searchHotel_fault element. Otherwise, send the value of the message element.

**Figure 26 An example Choose statement**

### If Statements

If statements enable you to specify a condition, and if the condition is met, then the specified mapping is output. When you chose this option, an If statement appears before the selected element, and you must place an XPath expression in the If statement that evaluates to a boolean. If the expression evaluates to true, the specified mapping is performed. If the expression evaluates to false, the mapping is not performed and no value is set for the item. Do not place an If statement around schema data components that are marked as required.

**Figure 27** shows an If statement. In this example, the returnDate schema element is optional. The returnDate element is surrounded by an If statement that evaluates whether the roundTrip element is true. If roundTrip is true, then the element is output, if roundTirp is false, the returnDate element is not output. The expression in the If statement is:

```
string($MessageData/ns:searchAirlineRequest/ns:parameters/ns0:searchAirline/roundTrip) = "true"
```
**For Each Statements**

For Each statements enable you to execute one or more statements once for each data element in a list. When you choose this option, a For Each statement appears before the selected data component, and you must place an XPath expression in the For Each statement that evaluates to a list of zero or more items. This is useful when you want to manipulate sequences or repeating elements.

Figure 28 shows a For Each statement. In this example, the requestor sends a list of ticker symbols and the stock exchanges on which they are traded. The mediation flow routes the request to different services for each stock exchange. The For Each statement takes the list of ticker symbols and executes the remaining statements once for each symbol in the list. The If statement examines the exchange element and outputs only the ticker symbols for the "NYSE" stock exchange.
Variables

Variables can be used in any XPath expression within your message panel. Choosing this option opens a dialog that enables you to specify the name of the variable. You can change the name of the variable at a later time by selecting the variable and clicking the **Show Edit Tab** button in the toolbar. The **Variable Name** field can be used to change the variable’s name.

The value of the variable is specified by supplying an XPath expression, either by mapping data from the context panel or by using XPath functions or constants. Once the variable’s contents have been supplied, the variable can be referenced within the scope that it has been defined. That is, you can reference a variable from within the same component or within sub-components of the component in which the variable is defined.

Adding a variable is useful if you perform the same computation repeatedly. You can refer to the results of the computation in several message elements instead of recreating the computation for each item.

Figure 29 shows adding a variable named `currPlusMarkup`. In this example, the variable uses the mediation flow property `USDtoYenXChangeRate` to get the value of the current exchange rate. That value is then multiplied by 1.02 to add a 2% markup. The variable can then be referenced in subsequent statements in the mapping.
Comments

Comments are used to supply information about your mappings. You can use comments as reminders or documentation for your application.

Comments are ignored by the software.
A mapping correlates data from the schema in the context panel with a data component in the message panel.

Several functions allow you to create and manage mappings:

<table>
<thead>
<tr>
<th>Function</th>
<th>Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a mapping</td>
<td>Drag and drop data components from the left-hand panel to the right-hand panel. The appropriate XPath expression is displayed in the <strong>XPath Expression</strong> field.</td>
</tr>
<tr>
<td>Add functions or constants to an XPath expression</td>
<td>Use the tabs at the bottom of the context panel (described in <strong>Data/Function Tabs</strong> on page 91).</td>
</tr>
<tr>
<td>Open a larger window in which to view or edit an XPath statement</td>
<td>Click an expression in the XPath Expression field, or click the <strong>Show Edit Tab</strong> button on the toolbar.</td>
</tr>
</tbody>
</table>

Lines appear between data components that are mapped to each other. The lines are blue when both components are visible, but the lines turn into a dashed green line when one or more mapped components are collapsed into its parent in the schema tree.

Data components in the message panel are initially displayed in italics. Italic text indicates that the components are hints to the potential mappings you can create.

Once you create a mapping for a data component, the hint changes from italics to non-italic font. Non-italic font indicates that the mapping is now an XSLT statement that transforms the data into the specified component. You can change a hint into a statement without performing a mapping by selecting a component in a message schema and dragging it past the dividing line between the left and right panels.

Data components on the right-hand side of a mapping can be either black or red. If the component is black, the XSLT statement for the component is valid and complete. If the component is red, that indicates the statement for the component is an error and must be repaired as described in the next section.
Repairing Incorrect Mappings

Any incorrect statements are displayed in red in the message panel. Errors can occur for a number of reasons:

- A required component has no statement and therefore must be specified.
- The message schema has changed, and existing statements may no longer be valid.
- The XPath formula for a component may contain an error.

Correct any errors before attempting to execute your mediation flow. If you hover the cursor over any red component name in the message panel, a pop up describing the error opens.

To help find potential problems in your mappings:

1. Click the Show Check and Repairs button on the toolbar. This button opens a dialog with all potential problems in the specified mappings.

2. Select the Fix checkbox for potential errors, and the software will attempt to automatically fix the problem.

Some potential problems in the Show Check and Repairs dialog cannot be fixed easily, and no check box for these items appears in the Fix column. For example, if a component expects a string and you supply a complex type, the corrective action to fix the problem is not clear. The problem cannot be automatically fixed. You must repair these items manually.

To delete mappings:

1. Select the parent parent or child component.

2. Click the Remove Mappings button in the toolbar.

   If a child component is selected, the component is returned to its original state and no mapping is specified.

   If a parent component is selected, mappings for all child components are also removed.

Mapping an Empty Complex Type

To map data of Empty Complex type, drag the empty complex object from the Target Operation Context pane on the right to the Mediation Operation Context pane on the left side, as shown in the following figure.
Chapter 6  Transforming Data in a Mediation Exchange
Using XPath

The Input tab uses XPath as the language for locating and manipulating data. XPath (XML Path Language) is an expression language developed by the World Wide Web Consortium (W3C) for addressing parts of XML documents. XPath also provides basic manipulation functions for strings, numbers, and booleans. To use XPath in the Input tab, you need only be familiar with the basic XPath concepts, but you might want to learn more about XPath when building complex expressions. For a complete description of XPath, refer to the XPath specification (which can be obtained from www.w3.org).

Addressing Schema Components

All data in the context and message panel is represented as an XML schema. The data can be a simple (strings, numbers, booleans, and so on) or a complex component. Complex components are structures that contain other schema components, either simple components or other complex components. Both simple and complex components can also repeat. That is, they can be lists that store more than one component of the given type.

XPath is used to specify which schema component you would like to refer to. For example, this schema may be available in the context panel:
The context panel of the example shows the schema available for a mediation operation. Three top-level items, each a root node in the context panel, are present: MediationFlowProperties, MessageContext, and MessageData. Each of these nodes has its own associated structure. MediationFlowProperties has a complex component named properties and MessageData has a complex component named searchHotelRequest.

References to a particular data item in any of these schema start with the root node and slashes (/) indicate a path to the data component. For example, the country element in the SearchHotel complex component that is in the parameters component would look like this in an XPath mapping field:

$MessageData/searchHotelRequest/parameters/searchHotel/country

The path starts with a dollar sign, then continues with node names using slashes, like a file or directory structure, until the location is named.

Some schema components must be prefixed with their namespace prefix. The prefix is automatically added to components that require this when dragging and dropping data in the XPath Expression field.

**Evaluation Context**

XPath also has a method for referencing relative paths from a particular node. If you have an evaluation context, or a particular starting node in a schema tree, you can specify the relative path to other elements in the tree.

For example, if your evaluation context is $MessageData/searchHotelRequest/parameters/searchHotel, you can reference the sub-items of ShipName without specifying the entire path. If you want to reference $MessageData/searchHotelRequest/parameters/searchHotel/country, the relative path would be ./country. The path is relative to the evaluation context — country is at the same level in the schema tree as the evaluation context.

**Search Predicates**

An XPath expression can have a search predicate. The search predicate is used to locate a specific element of a repeating schema item. For example, consider a schema where the $MessageData/searchReservations/todaysReservations item is a repeating element. If you want to select only the first item in the repeating element, you would specify this:

$MessageData/searchReservations/todaysReservations[1]

The [1] specifies the first element of a repeating item.
Sub-items can also be examined and used in a search predicate. For example, to select the element whose reservationId is equal to "3A54", you would specify:

$MessageData/searchReservations/todaysReservations[reservationId= "3A54"]

In the example above, the evaluation context of a predicate is set to the item containing the predicate. Therefore, reservationId is assumed to be within the todaysReservations component.

You can also use functions and expressions in the search predicate. For example, if you want to find all elements after the first, you would specify:

$MessageData/searchReservations/todaysReservations[position() > 1]

**Testing Boolean Values**

To test the value of a boolean node, you can use the `data()` function to obtain the value of the node. A common error in XPath functions is to supply a boolean node in a condition and expect that the condition will evaluate to true or false based on the value in the node. For example:

```xml
if
  ($MessageData/searchHotelRequest/parameters/searchHotel/nonSmoking)
then ...
```

The condition in the if statement above would return `true` when the nonSmoking component is present, regardless of whether the value of the component is true or false. To evaluate the value of a boolean element, use this expression:

```xml
if (data($MessageData/searchHotelRequest/parameters/searchHotel/nonSmoking)) then ...
```

You can also use the `string()` function to coerce the comparison to the string value of the Boolean node and then compare to the value of "true" or "false". For example:

```xml
string($MessageData/searchHotelRequest/parameters/searchHotel/nonSmoking) = "true"
```

**Comments**

You can add comments to XPath expressions using the XPath 2.0 syntax for comments. The syntax is:

```xml
{{-- <comment here> --}}
```

For example, this XPath expression contains a comment:

```xml
$MessageData/searchHotelRequest/parameters/searchHotel/country {-- returns the country --}
```
Transforming XML with Related Tasks

In addition to the Transform mediation task, ActiveMatrix provides tasks that enable you to manipulate XML data in text, binary, or tree formats.

**The Parse XML Task**

The Parse XML task is used when you have an XML document stored in a string or binary field. This task produces a tree representation of the XML that can be used by subsequent tasks in the mediation flow. You can pair the Parse XML task with the Render XML task to convert the parsed XML back into a string or binary field for transmission within a message. See Parse XML on page 188 for reference information about this task.

**The Render XML Task**

The Render XML task takes an XML tree for a specified schema and converts it to a string or binary element that contains the XML document. You can pair the Render XML task with the Parse XML task to convert the parsed XML back into a string or binary field for transmission within a message. See Render XML on page 195 for reference information about this task.

**The Validate XML Task**

You can use the Validate XML task to validate message data, a WSDL message, XML text, binary, or XML tree formats against a schema. The output of the Validate XML task is contributed to the mediation exchange, and can be used by downstream tasks. See Validate XML on page 214 for reference information about this task.

In addition to the Validate XML task, the message received by the mediation component can be validated using the VALIDATE_MESSAGE_DATA property that is added by default to mediation flows. See Validation, page 23 for more information.
Chapter 7  Querying a Database

This chapter describes using the Query Database task to look up information in a database by performing a SQL SELECT statement.

See Query Database on page 191 for reference information about this task.

Topics

- Overview of Query Database Tasks, page 112
- JDBC Resource Templates, page 113
- Registering a JDBC Driver, page 114
- Configuring a Query Database Task, page 116
Overview of Query Database Tasks

The Query Database task performs a SQL SELECT statement on a database. The task can specify three types of records:

- One or more tables in the FROM clause of the SELECT statement
- One or more columns to return in the SELECT list
- One or more conditions in the WHERE

You also have the option to specify the maximum number of rows to return.

The Query Database task can be used to look up data in a database table to enrich the data available in a mediation flow.

These are two usage scenarios:

- Store service names and namespaces for dynamically bound service references in a database table. You can then update the database table when a new service becomes available, and the mediation flow does not need to be changed to obtain the information about the new service.

- Use a database query to add information to an incoming request. For example, an incoming request may specify a US postal zip code, and a database query can be used to look up the city and state to add this information to the request.
JDBC Resource Templates

Resource templates are defined on the mediation flow and are used to specify a resource (such as a database connection) that can be used by one or more tasks in a mediation flow.

The property specified on the Properties tab of the mediation flow provides the database connection used for each Query Database task.

To connect to more than one database or use different user accounts, create one resource template for each database connection. Query Database tasks that use the same resource template will use the same database connection.

A JDBC resource template can be used to establish a connection to a database and obtain table and column information to complete the SELECT statement. This resource template is only used during design. When the mediation flow is used in a composite and deployed, the resource template is ignored.

Resource templates must be associated with JDBC resource templates at the component or composite level, and you can also override JDBC resources at deployment time in the ActiveMatrix Administrator interface.

See Composite Development for more information about resource templates and the Administrator interface.

To define a resource template:

1. Navigate to the Properties tab of a mediation flow by clicking on the canvas of a mediation flow in the editor window.
2. Click the + icon to add a new property.
   By default, the property name is specified as property\textsubscript{n} (where each newly added profile increments \textsubscript{n}). Specify a new name for the profile, if desired.
   The value in the Type column must be JDBC Resource Template. This value is read-only.
3. In the Value column, click the ellipsis (...) and choose a previously defined template from the Select JDBC Resource Template dialog box.
4. Click OK.
   The JDBC Resource Template is created and is ready for use by the Query Database task.
Configuring a JDBC Driver

Before it can be used, the JDBC driver referenced by the JDBC Resource Template must be configured.

To configure the JDBC driver:
Navigate to Window> Preferences> Data Management> Connectivity> Driver Definitions.

You can optionally specify a JDBC Resource Templates for use while creating Query Database tasks. JDBC Resource Templates define connections to databases. See Composite Development for more information.

Registering a JDBC Driver

To connect to a database at design time from within the Query Database task, you must first register the JDBC driver.

To register a database:
1. Navigate to the Data Source Explorer tab.
2. Right-click Database Connections and select New...
The Connection Profile pane displays.
3. Select the driver from the list of Connection Profile Types and click Next.
The Specify and Driver and Connection Details pane displays.
4. Click the New Driver Definition icon location to the right of the Drivers drop-down list.
The New Driver Definition dialog box displays.
5. Select the JDBC Driver in the Available driver templates list of the Name/Type tab.
6. Navigate to the Jar List tab.
The jar file for the selected database is listed in the list of driver files.
7. Select the JAR file (generated according to the selected JDBC driver) and click Remove JAR/Zip.
The JAR file is removed from the list of driver files.
8. Click Add JAR/Zip.
9. In the Select the file dialog box select the driver appropriate for your database, and click Open.
10. Click OK.
The database is now registered and is ready to be used within the Query Database task.
Configuring a Query Database Task

The following sections describe the configuration tabs of the Query Database task.

General Tab

On the General properties tab, you can specify a name and description. You must also select one of the resource templates defined for the mediation flow. See JDBC Resource Templates on page 113 for more information.

The Max Row Count field specifies the maximum number of rows to accept from the query results. For example, a positive integer of 1 returns only one row. The choice of Unlimited allows an unlimited number of rows in the result set.

The Query Timeout field specifies the timeout, in seconds, for a query statement to execute before an exception is thrown.

Query Tab

You use the Query tab to define the SELECT statement for the query.

If you specified a JDBC property in the mediation flow Properties tab, clicking the connection icon opens a connection and compares the table and column data with the metadata from the database. If the connection is not successful, an error notifies you of the reason.

Three lists enable you to select tables, input data, and output columns for use in the WHERE clause of your SELECT statement.
Input data is used in the WHERE clause of your SELECT statement. Use the add (+) and delete (x) icons to the right of each list to add and delete items from each list:

- When a database connection is present and valid, the + icons display information from the database for selecting tables and output columns.
- When no database connection is present, the + icons allow you to add items to each list, but you must name each item and specify a type if necessary.

Clicking the + and x icons on the Input table attempt an automatic update of the WHERE condition. If you have modified the WHERE condition, the delete might not update it and you must fix it manually.

Use the Where Condition field on the Query tab to edit the WHERE clause of the query. You can add an input variable to a condition by typing a question mark (?) in the condition. Each input variable appears in the mapper panel on the Input tab, and you can supply data from the mediation exchange for the input variable. For example, if you want to create a condition where you look up a ZIP code supplied in the input message, you can add the condition table.ZIP = ?. When you add a question mark into the WHERE clause, an input variable appears in the Input Data list. Supply a name for the input variable, then data from the mediation exchange can be mapped to the input variable.

Table join conditions are never automatically added to the WHERE clause. To specify any join conditions for your query, you must manually edit the WHERE clause.

The SQL Statement field displays a read-only version of the query you have specified. Table 12 lists the supported SQL types and how they map to XML. Length parameters are stripped from the SQP Type, and only the base type is used in the mapping. For example, char(12) becomes char.

<table>
<thead>
<tr>
<th>SQL/92 Data Type</th>
<th>XML Type Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>TINYINT</td>
<td>short</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>short</td>
</tr>
<tr>
<td>INTEGER</td>
<td>int</td>
</tr>
<tr>
<td>BIGINT</td>
<td>long</td>
</tr>
<tr>
<td>REAL</td>
<td>float</td>
</tr>
<tr>
<td>FLOAT</td>
<td>float</td>
</tr>
</tbody>
</table>
Vendor-specific types are cast to string. You can enable the mapper to automatically recognize these types in two ways:

- Force vendor-specific types to a compatible XML type using the mapper cast.
- Override the type that is retrieved from the database for the column to a similar SQL/92 type.

Binary or other complex data types such as JAVA_OBJECT are not supported.

### Input Tab

The **Input** tab is a mapping panel for mapping data from the mediation exchange to the input fields of this task. See Chapter 6, Transforming Data in a Mediation Exchange, on page 75 for more information about using a mapping panel.

### Output Tab

The **Output** tab is a read-only display of the output schema for this task. The output schema is determined by the output columns selected on the **Query** tab.

---

**Table 12  Supported SQL types and their mapping to XML**

<table>
<thead>
<tr>
<th>SQL/92 Data Type</th>
<th>XML Type Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOUBLE</td>
<td>double</td>
</tr>
<tr>
<td>CHAR</td>
<td>string</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>string</td>
</tr>
<tr>
<td>NCHAR</td>
<td>string (multi-byte)</td>
</tr>
<tr>
<td>NVARCHAR2</td>
<td>string (multi-byte)</td>
</tr>
<tr>
<td>DATE</td>
<td>date</td>
</tr>
<tr>
<td>TIME</td>
<td>time</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>dateTime</td>
</tr>
</tbody>
</table>

---
Test Tab

Use the Test tab to supply test data for values of input variables and test the query against the database associated with the specified JDBC resource template. To test the statement, a valid database connection must be present.

You can use a custom JDBC driver to test the database query. For information about configuring a custom JDBC driver, see Composite Development.

You must have a valid JDBC resource template associated with the shared resource profile used by this task. The JDBC shared resource is used only in the design environment.

Ensure that the JDBC resource template you use for testing in the design environment connects to a database that is similar to the database used when the project is put into production.
Chapter 8  Routing Requests Dynamically at Runtime

This chapter describes using dynamic binding to route incoming requests to target services.

Topics

- Overview of Dynamic Binding, page 122
- Configuring Dynamic Target Interfaces, page 125
- Setting the Dynamic Reference Task, page 127
- Configuring Dynamic References in Composite, page 132
- Creating and Deploying Composites Used By Dynamic Binding, page 133
Overview of Dynamic Binding

Dynamic binding enables you to route incoming requests to target services as they are needed at runtime.

As described in Mediation Components on page 45, target interfaces in a mediation flow correspond to component references in an ActiveMatrix composite. Typically, a component reference is wired to a composite reference that points to a service provider. This static binding is specified when the mediation component and composite are designed, and the service binding is hard-coded into the composite.

Dynamic binding allows components to supply a reference to the service provider when the deployable application archive (DAA) created from the composite is running.

Figure 30 shows the differences between static and dynamic binding in composite references.

Figure 30 Static and dynamic binding

The composite reference is statically bound to Service A. However, the dynamic composite reference can invoke Service B, Service C, or Service X without having to specify a static configuration at design time.
Dynamic references allow the component to specify which service to invoke. Therefore, new services can be started and a component can invoke those services without redesigning the composite and restarting the DAA created from the composite.

One example of using dynamic references is a set of services that return information for United States postal ZIP codes. The consumer sends a message to a mediation component containing the ZIP code. The service provider can implement a number of services for particular ZIP codes. When new ZIP codes are introduced, dynamic binding allows the service provider to create and start a new service for the new ZIP codes without changing any existing composites. Requests for information about new ZIP codes are handled without system downtime.

**Service Providers for Dynamic Composite References**

Dynamic composite references can refer only to bindings of type virtualization. That is, the service type in the provider composite cannot be JMS or SOAP. If your service provider uses the SOAP or JMS protocol, you can create a simple pass-through composite that passes the message to the ultimate service provider.

*Figure 31* shows a dynamic composite reference using a composite that implements a service and also using a pass-through composite for referencing a SOAP service.

*Figure 31  Service providers and pass-through composites*
Referring to Service Providers

The component implementation determines the service that is invoked for a dynamic reference. To specify the service, the implementation supplies the application name and service name. ActiveMatrix resolves the application name and service name to the correct running service.

The service name is the name specified for the promoted service in the composite.

Configuring Dynamic Binding

This topic provides an outline of the steps for configuring dynamic binding. (The remaining sections in this chapter describe how to perform each of the steps to configure dynamic binding.)

To configure applications to use dynamic binding, these procedures are required:

- **Configure dynamic target interfaces** (see page 125).
  - Add target interfaces to a mediation component.
  - Specify that the interfaces are dynamic.
- **Set the dynamic reference task** (see page 127).
  - Add the Set Dynamic reference task to your mediation path.
- **Configure dynamic references in the composite** (see page 132).
  - Create a component reference and specify that it is wired by implementation.
  - Wire the dynamic component reference of the mediation component to the dynamic composite reference.
- **Create and deploy composites used by dynamic binding** (see page 133).
  - Create composites with service virtualization that either implement the service or pass-through to a SOAP or JMS service.
Configuring Dynamic Target Interfaces

Target interfaces in a mediation flow can be specified as either static or dynamic. By default, a target interface is static. The target interface corresponds to a component reference that is wired to a composite reference in a composite. Dynamic target interfaces correspond to dynamic component references that are wired to dynamic composite references.

Marking a Target Interface as Dynamic

You can specify whether a target interface is static or dynamic. By default, target interfaces are static.

Figure 32 shows setting the **Wired by Implementation** field for a target interface and shows the color and icon of the title bars of dynamic interfaces.

Figure 32  Dynamic and static target interfaces

To mark a target interface as dynamic:

1. Go to the **General** tab of the Properties view of the target interface.
2. Select the **Wired by Implementation** field.
   
   The title bars of the target interfaces is shaded yellow, and a lightning bolt icon is added.
Using Pattern Variables

Once a target interface is marked as dynamic, the application name and service name must be supplied in the mediation flow. You can optionally specify pattern variables to aid in the mapping of data in the mediation exchange to create the appropriate application name and service name.

For example, you can have six operations in your dynamic target interface. All service providers that are referenced use the same application name, and service name follows the pattern service data, where data is a variable portion of the service name. The value of data is supplied in an incoming message.

Instead of providing a mapping for each of the six operations, you can specify one pattern variable to represent the variable portion of the service name, and then create one mapping for all operations on the Pattern Map tab of the Properties view of the target interface. Figure 33 shows the Pattern Map tab of this example.

Figure 33   An example of pattern variables

To supply the value of the pattern variable, you can perform one simple mapping in a Set Dynamic Reference task on the path for each mediation operation.

See Setting the Dynamic Reference Task on page 127 for more information about the Set Dynamic Reference task.
Setting the Dynamic Reference Task

The Set Dynamic Reference task provides the values needed for resolving a service provider in a dynamic target interface. Each Set Dynamic Reference task sets the value of the service provider for the specified dynamic target interface. The value is then used by the next item that refers to a dynamic target interface — either the end of the mediation path points to a dynamic target interface, or an Invoke task invokes an operation on a dynamic target interface.

You might need more than one Set Dynamic Reference task along a mediation path in these situations:

- The target interface is marked as dynamic and there is an Invoke task on the path configured invoke a different dynamic target interface.
- More than one Invoke tasks are on a path, and each task invokes a different dynamic target interface.
- You want to Invoke the same operation on a dynamic target interface more than once, and each time you want to set the dynamic reference to a different value.

Figure 34 shows the usage cases of the Set Dynamic Reference task.

A grey hint line appears between the Set Dynamic Reference task and the corresponding dynamic target interface. A yellow hint line appears between a Set Dynamic Reference task and the corresponding Invoke task when you select a Set Dynamic Reference task in the mediation flow. In Figure 34, the diagrams have been changed to show all hint lines, even though only the hint lines for the selected task can be viewed in the mediation editor.

The Set Dynamic Reference task is typically used on input mediation paths. It can be used on output or fault paths when an Invoke task that invokes an operation on a dynamic target interface appears on an output or fault path.
Figure 34  Usage cases for Set Dynamic Reference task

One dynamic target interface and one Set Dynamic Reference task.

Two dynamic target interfaces and two Set Dynamic Reference task. The first Set Dynamic Reference task sets the reference for the target operation. The second Set Dynamic Reference task sets the reference for the Invoke task.

Two Invoke tasks, each executing different operations on the same interface. The first Set Dynamic Reference task sets the reference for the first Invoke task. The second Set Dynamic Reference task sets the reference for the second Invoke task. A different service provider can be invoked by each Invoke task.

Configuring the General Tab

Use the General tab to specify the name and description of the Set Dynamic Reference task, and to set the target interface and endpoint reference mechanism:

- **Dynamic Target Interface** specifies the name of the dynamic target interface for which this task is supplying the service name and namespace. By default, this field is automatically set to the dynamic target interface at the other end of the path. If there is an Invoke task on the path, this field may be set by default to
the first dynamic target interface in the target interface list. You might need to set this field when the default choice does not match the dynamic target interface that you want to set.

- **Endpoint Reference Mechanism** Select the mechanism to use for setting the application and service name:

  ![Endpoint Reference Mechanism](image)

  **Application & Service Name** Select to supply the application name and service name. This option requires two inputs for mapping on the Input tab—ApplicationName and ServiceName.

  **Application & Service URI** Select to supply the exact URI of the endpoint. This is useful if, for example, someone sends you the URI—you can simply copy and paste it into the ApplicationServiceURI parameter on the Input tab.

  **URI** Select this option to specify the URI in the Input tab:

  **Pattern Variables** Select to use pattern variables from the dynamic target interface. This is useful if several operations in a dynamic target interface use a similar pattern for the application name and service name. You can specify the mapping once on the dynamic target interface and use variables to supply the variable portion. The variables you create on the dynamic target interface appear in the Input tab when this option is selected.

  By default, the **Endpoint Mechanism** field is set to Application & Service Name when the dynamic target interface has no pattern variables.

  If the dynamic target interface has pattern variables, the **Endpoint Mechanism** field is set to Pattern Variables by default.

**Specifying the Input**

When the **Endpoint Reference Mechanism** field on the **General** tab is set to Application & Service Name, the input elements for this task are ApplicationName and ServiceName. Any value you specify for these input elements override the value specified on the **Pattern Map** tab of the specified dynamic target interface.
When the **Endpoint Reference Mechanism** field on the **General** tab is set to Pattern Variables, then the pattern variables defined on the specified dynamic target interface are the input elements. This enables you to specify simple mappings of data from the mediation exchange to the variable values. The variable values are then passed to the mapping supplied on the **Pattern Map** tab of the dynamic target interface.

Use any data available in the mediation exchange on the left side of the mapper panel to provide data to the input values. See Chapter 6, *Transforming Data in a Mediation Exchange*, on page 75 for more information about mapping data in the Input tag.

The content of the **Input** tab depends on which Endpoint Reference Mechanism you select on the **General** tab—Application & Service Name, Application & Service URI, URI, or Pattern Variables:

- **Application & Service Name** When the **Endpoint Reference Mechanism** field on the **General** tab is set to Application & Service Name, the input elements for the Set Dynamic Reference task are ApplicationName and ServiceName:
  - ApplicationName refers to the application name provided during deployment of a composite, to uniquely identify an instance of an application template.
  - ServiceName is the name of the composite service that is contained in the target composite.

- **Application & Service URI** When the **Endpoint Reference Mechanism** field on the **General** tab is set to Application & Service URI, the input element for the Set Dynamic Reference task is ApplicationServiceURI.
  
The data type of the **ApplicationServiceURI** input field is a URI of the format 

  urn:amx:EnvironmentName/ApplicationName#service(ServiceName)

  The variables **EnvironmentName**, **ApplicationName**, and **ServiceName** refer to the environment and service that are being invoked:
  - **EnvironmentName** is the name of the ActiveMatrix environment that contains the target service.
  - **ApplicationName** refers to the application name that is provided during deployment of a composite, to uniquely identify an instance of an application template.
  - **ServiceName** is the name of the composite service that is contained in the target composite.
- **URI** When the **Endpoint Reference Mechanism** field on the **General** tab is set to URI, the input elements for the Set Dynamic Reference task are URI:
  - SOAP over HTTP
  - SOAP over JMS
  - ActiveMatrix Service Virtualization

- **Pattern Variables** When the **Endpoint Reference Mechanism** field on the **General** tab is set to Pattern Variables, the pattern variables that are defined on the specified dynamic target interface are the input elements.
  
  This enables you to specify simple mappings of data from the mediation exchange to the variable values. The variable values are then passed to the mapping supplied on the **Pattern Map** tab of the dynamic target interface. You can use any data available in the mediation exchange on the left side of the mapper panel to provide data to the input values.
Configuring Dynamic References in Composite

Dynamic target interfaces in a mediation flow correspond to dynamic component references in mediation components that use the mediation flow as an implementation. Dynamic component references must be wired to dynamic composite references in an ActiveMatrix composite.

See Composite Development for more information about creating and configuring composite references.

To set a composite references to be of type dynamic:

1. Open the General tab of the promoted reference.
2. In the Advanced section, select the Wired by Implementation field.
   The references and services must be promoted to the composite level for this setting to take effect.
Creating and Deploying Composites Used By Dynamic Binding

TIBCO ActiveMatrix resolves the application and service names provided by a component to a running application that contains the corresponding service of binding type virtualization. The composite with the corresponding service can implement a service or it can pass through to another service using the SOAP or JMS protocol as described in Service Providers for Dynamic Composite References on page 123.

You can create composites using the Composite Editor, or you can use the automatic mechanism in the Mediation Editor to create composite services that a dynamic target interface can use.

To use the automatic mechanism in the Mediation Editor:

Once you complete these steps, you can configure the remainder of your service provider.

1. Click the down-arrow icon in the title bar of a dynamic target interface in a mediation flow and select Create Dynamic Provider from the menu.

This Create Dynamic Provider dialog opens:

2. Specify the Service Name, the Namespace and the Workspace Location in the fields provided.

3. Click the Browse button next to the Workspace location field to locate the project and folder in your workspace where you want to place the composite.

4. Click OK.

The provider composite created with the wizard is configured with a service with the specified name and namespace. The port type and WSDL location for the service are set to the target interface in the mediation flow.
Chapter 9  

Repeating to Messages

This chapter describes using the Generate Reply and Handle Reply tasks to send reply messages without invoking target operations.

Topics

- Overview of Reply Messages, page 136
- Working with Reply Messages, page 138
Overview of Reply Messages

In a typical mediation flow for an operation with an in-out message exchange pattern, incoming messages travel along the input path until the message is delivered to the target operation or until a fault is encountered.

In some situations, you might want to send a reply message to the consumer without invoking the target operation. For example, an operation might return the name of the target service. The mediation flow already has the target service name, so you can improve performance and return that information without additional network traffic to the target service.

Another example is a mediation flow with a route task for processing incoming requests. Your mediation flow might return an unchangeable message for one or more routing cases. In that case, you can reply to the consumer without invoking the target service.

You can place the Generate Reply task on an input path to terminate the path and pass control to the output path of the mediation flow. You must map the output message in the **Generate Reply Input** tab, so that the output message is created in the task.

On the output path, the Handle Reply task intercepts messages from any Generate Reply tasks on the input path and starts the mediation reply path for processing the reply message before it is sent to the consumer.

**Figure 35** shows the operation of the Generate Reply and Handle Reply tasks. In this example, a mediation flow for the `createUser` operation first invokes the `queryUser` operation to determine if the user exists. If the user does not exist, the message is delivered to the `createUser` target operation. If the user already exists, the Generate Reply task is used to return a message notifying the consumer that the user already exists.
Placing a Generate Reply task in the Input path automatically creates a mediation reply path with a Handle Reply task. The same Handle Reply task performs all Generate Reply tasks in the Input path.

Figure 35  Sending a reply message
Working with Reply Messages

The Generate Reply task terminates an input path before reaching a target operation. However, you can have more than one Generate Reply task on an input path when a route task splits the input path into multiple sub-paths. One or more sub-paths can end in a Generate Reply task. Generate Reply tasks are executed based on how they are configured in the input flow paths. The Handle Reply task is on the output flow.

*Figure 36  Setting the reply message*

After a Generate Reply task is executed, control is passed to the Handle Reply task on the output path. One Handle Reply task accepts reply messages from any Generate Reply task on the input path. The Handle Reply task starts the mediation reply path. Optionally, you can place tasks on the mediation reply path to perform additional processing before the reply message is sent to the consumer. The Handle Reply task and the mediation reply path are automatically placed into the mediation flow when a Generate Reply task is placed on the input path.

*Figure 37* shows the output path containing a Handle Reply task and the mediation reply path.
Figure 37  Handling the reply message

One Handle Reply task handles reply messages from any Generate Reply task. The Handle Reply task starts the mediation reply path. Optionally, you can place other tasks on the mediation reply task to execute before the reply is sent to the consumer.
Chapter 10  Fault Processing in a Mediation Flow

This chapter describes fault processing, errors that occur during processing.

Topics

- Overview of Faults, page 142
- Throwing Faults in Mediation Flows, page 145
- Working with Fault Paths, page 147
- Sending Faults to the Invoker, page 150
Overview of Faults

The Mediation Flow Editor enables you to specify a fault path for processing to occur when a fault is encountered. Faults are errors that can occur at any point along the mediation path. Faults can be thrown by the target service while processing messages. Faults can also be explicitly thrown during a mediation flow to specify that an unhandled case has occurred.

Invoke tasks, like any other task, can generate faults. If an operation referenced by an Invoke activity declares faults, those faults can be caught and processed in the fault path. For example, if the operation declares that it can throw FaultA and FaultB, these faults appear in the Fault Path as faults that can be caught and processed.

**Figure 38  Examples of where faults can occur**

- faults can occur when receiving the message and creating the mediation flow context
- faults can occur when executing tasks in input or output or fault paths of the mediation flow
- faults can occur when executing the target operation
- faults can be explicitly thrown with the Throw Fault activity

When a fault is encountered, processing of the current path is immediately halted, and control is passed to the fault path. The fault path enables you to catch explicitly declared faults or unhandled faults. However, if a fault is encountered when receiving the message, the mediation flow has not yet started, so the fault is immediately returned to the sender and no fault processing can be done in the mediation flow.

**Figure 39** shows a fault path.
By default, each target operation has one Catch Fault task with sub-paths:

- A sub-path for each declared fault that can be thrown by the operation
- One sub-path to handle any undeclared faults
- One sub-path for timeout faults.

There is also one Catch Fault task for faults that occur during processing of the mediation flow.

You can configure each Catch Fault task to have fewer sub-paths, if desired. When you remove sub-paths from a Catch Fault task, the Catch All path is automatically added to catch any faults where there is no specific sub-path for fault handling.

Each sub-path from each Catch Fault task leads to a Send Fault task. The Send Fault task sends a fault back to the original sender of the message. By default, the Send Fault task is configured to send the specific fault caught by the sub-path. You can configure the Send Fault task on a either a target or mediation fault sub-path to send either a generic UndeclaredFault or one of the specific fault messages defined on the mediation operation.
When the fault sent by the Send Fault task does not match the fault caught by the sub-path, a Transform activity is required to transform the fault message into the required format. For faults on the mediation fault path, the Transform activities are added by default, but if you change the configuration of the Send Fault or Catch Fault tasks, you must provide the correct Transform task as well.

You can place mediation tasks along the sub-paths between the Catch Fault activities and Send Fault activities to perform post-fault processing before the fault is returned to the original message sender.

For more information about how to configure the Catch Fault and Send Fault tasks, see Working with Fault Paths on page 147, Catch Fault on page 176, and Send Fault on page 201.

When faults are encountered while processing tasks in a mediation flow, the execution of the path is terminated, and the control sent to the mediation fault path. This includes faults that occur when processing tasks on any of the following paths:

- Mediation Input Path
- Mediation Output Path
- Mediation Reply Path
- Mediation Target Fault Path

When a fault is encountered on the Mediation Fault Path, the path terminates and a fault is sent to the consumer.
Throwing Faults in Mediation Flows

The Throw Fault mediation task enables you to explicitly throw a fault during processing on the input path of a mediation flow. This is useful in two situations:

- You want to deprecate a mediation operation, and therefore a fault is sent to all clients that request that operation.
- You want to specify routing cases where a fault is sent.

For example, if a loan processing application cannot process loans over $5,000,000, then you would configure a routing case for the loan request operation to examine the loan amount and place a Throw Fault task on the sub-path for the case where the loan amount was over $5,000,000.

The Throw Fault task enables you to browse through available service descriptors and select messages from the service to send as the fault message. You also can select which MediationTaskFault message to send. If you have more than one Throw Fault task and you want to perform specific processing for each task, configure each task to send a specific message.

Figure 40 shows how to configure the Throw Fault task to send a specific message.
Figure 40  Example of configuring the Throw Fault task

1. On the General tab, click Browse to select a service descriptor containing the fault message to send:

2. On the Select WSDL Message dialog, select the WSDL file in the Matching Resources field.

3. The Throw Fault activity is configured to throw the message, and you can navigate to the WSDL by clicking the WSDL Location field label:
Working with Fault Paths

Fault paths enable you to specify tasks to perform when a fault is thrown. To view the fault path for the currently selected mediation operation, use the **Show Fault Direction** button at the top of the mediation paths area of the mediation flow editor. There is one Catch Fault task for each target operation in the mediation flow, and one Catch Fault task for faults encountered while processing the mediation flow.

Catching Faults From the Target Operation

Each target operation in a mediation flow has a Catch Fault task that catches faults thrown by the target operation. The faults can be either explicitly defined faults in the target operation’s service description, or they can be unhandled exceptions encountered during processing (for example, a `NullPointerException`).

The default Catch Fault task for a target operation has sub-paths for each declared fault in the target operation and one sub-path each for a time out fault and any undeclared faults. You can place mediation tasks on each sub-path to perform any post-fault processing for each fault.

To specify the same processing for multiple faults, you can configure the Catch Fault task to have fewer sub-paths by unselecting the **Catch and Handle** field for the fault. When you eliminate one or more sub-paths, the Catch All sub-path is required, and it is automatically enabled. Any faults that do not have a defined sub-path are sent to the Catch All sub-path.

**Figure 41** shows the **General** tab of the Catch Fault task that allows you to configure the sub-paths for the faults to catch.

**Figure 41  Configuring a Catch Fault task**
In the example Catch fault task, one fault message is defined on the target operation named searchAirline_faultMsg. The Catch Fault task also has the following faults listed for all target operations:

- **UndeclaredFault** Catches any undeclared faults encountered while executing the target operation.
- **Catch All** Catches all remaining faults that are not explicitly defined. By default, this option is cleared. This option becomes required and is automatically selected if you clear the **Catch and Handle** field for any other faults.
- **Timeout Fault** Catches any timeouts encountered while executing the target operation.

When you select the check box in the Catch and Handle column for a fault, the Catch All sub-path is automatically added. Figure 42 shows the fault path that results when the **Catch and Handle** check box is cleared for searchAirline_faultMsg, Undeclared Fault, and TimeoutFault.

**Figure 42  Removing specific faults from the target operation fault path**

---

**Catching Faults From the Mediation Flow**

One Catch Fault task catches faults encountered while processing the mediation flow. Faults in a mediation flow can occur in these situations:

- An explicit fault is thrown with the Throw Fault task. This task can either throw the MediationTaskFault message or it can be configured to throw a different message defined in a service descriptor in the project.
- A mediation task throws the declared MediationTaskFault fault during processing (this also applies to tasks on target fault paths).
- An undeclared exception occurs during mediation processing. In this case, the MediationTaskRuntimeFault is thrown.
Figure 43 shows the Catch Fault task for the mediation flow. In this example, the mediation flow has a Throw Fault task that throws the `searchHotel_faultMsg` fault, and the `MediationTaskFault`, `MediationTaskRuntimeFault`, and catch all options are present in all Catch Fault tasks for mediation flows.

By default, the `MediationTaskFault` and `MediationTaskRuntimeFault` sub-paths are configured with Transform tasks that transform the caught fault into an `UndeclaredFault` message. If you check the Catch and Handle field for any faults declared on the target operation, you must configure the corresponding Send Fault task and provide any required transformations by adding a Transform task to the sub-path, if necessary.

**Catching All Faults**

If you choose not to catch specific faults from the target operation or the mediation flow, the Catch All fault option remains selected. In this case, the Fault to Send field of the Send Fault task contains an option Original Fault That Was Thrown. When this option is selected, all encountered faults are passed on the caller as they occur without any transformation.
Sending Faults to the Invoker

The Send Fault task sends a fault message back to the original process that invoked the mediation operation. You can configure the Send Fault task to specify what fault message to send:

- One of the fault messages declared on the mediation operation
- The UndeclaredFault message

To select what fault message to send, use the drop-down list in the Fault to Send field on the General tab of the Send Fault task.

Once you specify the fault message to return, you may need to place a Transform task on the fault path to convert the message sent by the Catch Fault task to the format of the fault message you are returning. The message panel of the Transform task on a fault path is labeled Mediation Fault Message, and the schema of the fault message matches the schema of the message specified in the Send Fault task on the path.
Chapter 11  Custom Mediation Tasks

This chapter explains how to define and deploy custom mediation tasks.

Topics

- Overview, page 152
- Migrating Custom Mediation Tasks, page 153
- Support Files, page 156
- Creating the Model Plug-in, page 157
- Creating the UI Plug-in, page 162
- Creating the Runtime Plug-in, page 165
- Writing Custom Mediation Code, page 168
- Packaging Custom Mediation Tasks, page 172
Overview

Custom mediation tasks are user-defined mediation tasks written to perform specific mediation functions.
Migrating Custom Mediation Tasks

If you created and deployed custom mediation tasks in earlier versions of ActiveMatrix, and you want to use them with this version, you must migrate them to ActiveMatrix 3.x.

To migrate custom mediation tasks you created in earlier versions of ActiveMatrix:

1. Import your 2.x model and UI plug-ins into your workspace.
2. The Model plug-in uses import packages. Replace all Required-Bundles with:
   com.tibco.amsb.core.mediation.model.report.impl
   com.tibco.amsb.core.componentType
   com.tibco.amsb.core.mediationbpe1.custom.task.persistence
   com.tibco.amsb.core.mediationextmodel
   com.tibco.amsb.core.mediation.model.ext.internal
   com.tibco.amsb.core.querymodel
   com.tibco.amsb.core.task.model.ext
   com.tibco.amsb.core.task.model.report
   com.tibco.amsb.core.taskdescriptormodel
   com.tibco.amsb.core.task.common.validation
   com.tibco.amsb.core.task.validation.constraints
   org.eclipse.core.runtime
   org.eclipse.bpel.model
   org.eclipse.emf.common
   org.eclipse.emf.ecore
   org.eclipse.emf.ecore.xmi
   org.eclipse.emf.ecore.impl
   org.eclipse.emf.validation
   org.eclipse.emf.common.util
   org.eclipse.emf.ecore.ecore.plugin
   org.eclipse.emf.common.notify
   org.eclipse.emf.common.notify.impl
   org.eclipse.emf.ecore.ecore.xmi.impl
   org.eclipse.wst.wsdl
   org.eclipse.xsd

3. Delete the methods getStaticInputTypeSchemaStream() and getStaticOutputTypeSchemaStream() methods from class <TaskName>MediationTaskReport.java.

4. Delete the class <TaskName>MediationTaskValidator.java after plugging out the validation code as the 3.x validation is based on EMF model constraints. The CMT developer must migrate the existing validation to EMF model constraints.

   Remove its extension entry in plugin.xml by deleting
   com.tibco.amsb.core.mediation.model.ext.MediationTaskValidation Extension entry.
5. Update the UI plug-ins.
   a. Update the following dependencies from:
      com.tibco.amsb.core.mediation.model,
      com.tibco.amsb.core.mediation.model.report,
      com.tibco.amsb.core.mediation.model.ext
      to:
      com.tibco.amsb.core.mediation.model.eclipse,
      com.tibco.amsb.core.mediation.model.report.eclipse,
      com.tibco.amsb.core.mediation.model.ext.eclipse
   b. Add the dependency: com.tibco.xml.mapper.schema.emfapi.
6. Re-compile the plug-ins by doing a clean build.
7. Re-create your runtime plug-in, and port your custom code using the Mediation API.
   See Creating the Runtime Plug-in on page 165 for more information.
8. Re-create the DAA in the SOA project that is created. See Deploying Custom Mediation Tasks on page 172 for more information.
9. Install the custom mediation task. See Packaging Custom Mediation Tasks on page 172 for more information.
Eclipse Plug-ins

A custom mediation task consists of three Eclipse plug-ins as described in Table 13.

Table 13  Custom mediation task plug-ins

<table>
<thead>
<tr>
<th>Plug-in</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model</td>
<td>The basis of automatic code generation for the design and runtime environments. The model contains attribute-value pairs that can be used in both environments. This plug-in consists of metadata based on the Eclipse Modeling Framework (EMF). This plug-in is used in both the design and runtime environments. For more information, see Creating the Model Plug-in on page 157.</td>
</tr>
<tr>
<td>UI</td>
<td>The user interface code and icons. This plug-in has extension points for the Properties view and the Mediation Palette in the Mediation Flow Editor. This plug-in refers to the model and is used in the design environment. For more information, see Creating the UI Plug-in on page 162.</td>
</tr>
<tr>
<td>Runtime</td>
<td>The Java code that performs the mediation logic. This plug-in refers to the model and is used in the runtime environment. For more information, see Creating the Runtime Plug-in on page 165.</td>
</tr>
</tbody>
</table>
Support Files

A custom mediation task might depend on support files such as:

- Schema files that describe the schemas of messages.
- Graphics files that are used as icons for the custom mediation task. Table 14 describes the icons that can be used. The icon formats can be GIF, JPEG, or PNG formats.

Table 14  Icons for custom mediation tasks

<table>
<thead>
<tr>
<th>Icon</th>
<th>Where Displayed</th>
<th>Recommended Dimensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>Mediation Palette: When the <strong>Use Large Icons</strong> option is not selected&lt;br&gt;Paths: When <strong>Small Icons</strong> is selected in the preferences</td>
<td>16 x 16 pixels&lt;br&gt;Default icon:</td>
</tr>
<tr>
<td>Large</td>
<td>Mediation Palette: When the <strong>Use Large Icons</strong> option is selected&lt;br&gt;Paths: When <strong>Large Icons</strong> is selected in the preferences</td>
<td>32 x 32 pixels&lt;br&gt;Default icon:</td>
</tr>
</tbody>
</table>

In the palette, icons appear on a light gray background. On the canvas, icons appear on a yellow gradient. For this reason, consider using a combination of hard edges (rather than anti-aliasing) and transparency when designing icons.
Creating the Model Plug-in

To create the model plug-in for the custom mediation task:

1. Run TIBCO Business Studio from the Start menu. For example, select **Start > All Programs > TIBCO_HOME > TIBCO Business Studio N.N > TIBCO Business Studio**.

2. Select **File > New > Project....**

3. In the New Project dialog under **Plug-in Development**, select **Plug-in Project** and click **Next**.

4. Specify a name for the project that reflects the mediation task name and that identifies this as the model plug-in. For example, type LookupTaskModel.

5. Accept all other defaults and click **Next**.
6. On the Plug-in Content page, locate the **Plug-in Options** group and clear these options:

   — Generate an activator, a Java class that controls the life cycle of the plug-in.

   — This plug-in will make contributions to the UI.

7. Accept all other defaults and click **Next**.

8. On the Templates page, select **Mediation Task Model Wizard** and click **Next**.
9. On the Mediation Task Model page, for the **Mediation Task Model Name** replace the string within the brackets with another of your choice. This prefix will be used for the Mediation Task Names for the UI and Runtime plug-ins.

This figure highlights the string to replace:

10. Specify the **Java Package Name** for the model plug-in for the custom mediation task. Click **Next**.

   By default, the Java package name is the same as the project name.
11. (Optional) Select a schema element for the custom mediation task input/output in the Input/Output Schema Selection dialog.
   a. Type the name of the schema.
   b. Click **Browse** to see all the schemas in the workspace, or click **Create** to create a new schema:

   ![Input/Output Schema Selection dialog](image)

   Clicking **Create** opens the Simplified Schema Editor. See Creating Simple Schemas on page 39 for information about using the Simplified Schema Editor.

12. Specify model attributes that the custom mediation task will use. Model attributes can be given values for each instance of the task, by specifying the values on the **General** tab in the Properties view for the task.
   - To add an attribute, click **Add**. Edit the attribute name and add default values for the attribute. Click the **Types** cell to select the attribute type.
   - To remove an attribute, highlight the row for the attribute by clicking in one of the cells on the row, and click **Remove**.
This following figure shows an example of a new model attribute:

13. Click **Finish**.

14. TIBCO Business Studio opens the Open Associated Perspective dialog, which asks if you want to open the Plug-in Development perspective.

   Optionally, check the check box **Remember my decision**. Select **Yes**. TIBCO Business Studio opens the model plug-in and the Plug-in Development perspective.

   The model plug-in for the custom mediation task is created.
Creating the UI Plug-in

Before you begin, close your runtime plug-in project.

To create the UI plug-in for the custom mediation task:

1. Close the RT project.
3. Select File > New > Project....
4. In the New Project dialog under Plug-in Development, select Plug-in Project and click Next.
5. Specify a name for the project that reflects the mediation task name and that identifies this as the UI plug-in—for example, LookupTaskUI.

Accept all other defaults and click Next.
6. On the Plug-in Content page, accept all defaults and click Next.

7. On the Templates page, select Mediation Task UI Wizard and click Next.

8. In the Mediation Task Model Selection Section, choose the mediation task model plug-in and click Next.

9. On the mediation Task UI page, the prefix that was chosen for the Mediation Task Name for the Model appears. Accept the default or specify a new one.

10. Specify the Java Package Name that will be used for the UI plug-in for the custom mediation task, or accept the default value. Click Next.

    By default, the Java package name is the same as the project name.

11. Specify the location of the small icon for the custom mediation task. The location should be the complete path to the file on your local hard drive. You can click Browse to browse to locate and select the file.

12. Specify the location of the large icon for the custom mediation task. The location should be the complete path to the file on your local hard drive. You can click Browse to browse to locate and select the file.

13. Click Finish.

The UI plug-in for the custom mediation task is created.

You can now install the custom mediation tasks. See Installing Custom Mediation Tasks.
In case you see compilation errors switch the Target Platform to TIBCO ActiveMatrix SOA Studio. See Composite Development for information on switching the Target Platform.
Creating the Runtime Plug-in

Before you begin, close your UI plug-in project and your feature project.

To create the runtime plug-in for the custom mediation task:

1. Close the feature/UI project that was created when the custom mediation tasks were installed.


3. Select File > New > Project....

4. In the New Project dialog under Plug-in Development, select Plug-in Project and click Next.

5. Specify a name for the project that reflects the mediation task name and that identifies this as the runtime plug-in, for example, LookupTaskRuntime:

Accept all other defaults and click Next.
6. On the Plug-in Content page, locate the **Plug-in Options** group and deselect these options:

   — Generate an activator, a Java class that controls the life cycle of the plug-in.
   
   — This plug-in will make contributions to the UI

7. Accept all other defaults and click **Next**.

8. On the Templates page of the wizard, select **Mediation Task Runtime Wizard** and click **Next**.

9. On the Mediation Task Model Selection Section, choose the mediation task model plug-in and click **Next**.

10. Specify the **Mediation Task Name**. This is a unique name that reflects the nature of the custom mediation task, for example, LookupTask.

11. Specify the **Java Package Name** for the runtime plug-in for the custom mediation task. Click **Next**.

   By default, the Java package name is the same as the project name.

12. Click **Finish**.

13. TIBCO Business Studio opens the Open Associated Perspective dialog, which asks if you want to open the Plug-in Development perspective.

   Optionally, check the check box **Remember my decision**. Click **Yes**. TIBCO Business Studio opens the runtime plug-in and the Plug-in Development perspective.

Along with the Runtime plug-in, a SOA Project `<runtime plug-in project name>.deploy.soa` is created as shown in the following figure:

![Diagram of SOA Project structure](image)

This SOA project contains the Custom Mediation Task Extension component that refers to the runtime plug-in.

Your custom code is written in `<runtime plug-in project name>/src/<runtime plug-in project name><custom mediation task name>RT.java`. 
Do not update or delete the generated `<runtime plug-in project name>.apt.composite`. This composite is generated for the sole purpose of packaging the custom tasks plug-ins into deployable artifacts.

If you see compilation errors, switch the Target Platform to TIBCO ActiveMatrix Runtime. See *Composite Development* for information on switching the Target Platform.
Writing Custom Mediation Code

Custom mediation code performs operations on MediationExchange in paths, and on specific elements of the message and path contexts.

Before modifying the Task EMF Model, import the required plug-ins:
1. Make sure the target platform is set to TIBCO ActiveMatrix Runtime.
3. Click Next.
4. Make sure the check box for importing from the active target platform setting is selected.
5. Click Next.
6. Import the following plug-ins by selecting them in the Plug-ins and Fragments Found table and clicking Add -->:
   - com.tibco.amsb.core.model
   - com.tibco.amsb.core.mediation.model.ext
7. Click Finish.

Accessing Task Input/Output Schema

To access the input or output element declaration at runtime, you must initialize a mediation task report object MediationTaskNameReport. This object has the accessor methods to get the input or the output element declaration as XSDElementDeclaration.

This code shows how to get the input or the output element schemas:

```java
public void init() throws TaskLifeCycleFault { }

public void destroy() throws TaskLifeCycleFault { }
public N execute(final N input, final Exchange<N> exchange) throws TaskFault {
    <TaskName>MediationTask task =
    this.getContext().getTaskConfiguration();
    <TaskName>MediationTaskReport report = new
    <TaskName>MediationTaskReport(task);

    //Task input type as schema element declaration
    XSDElementDeclaration inputType = report.getCustomInputType();

    //Task output type as schema element declaration
    XSDElementDeclaration outputType = report.getCustomOutputType();
```
return exchange.getMessageData();

The return type of the execute method in a custom mediation task’s runtime class that extends MediationTaskRT must be an instance of the output schema defined for the task. If no output schema is defined, output defaults to message data mediationExchange.getMessageData().

Modifying the Mediation Task Data

The execute method of the mediation task runtime class has MediationExchange and the task input as its arguments. The mediation exchange holds the mediation message and the exchange variable as a generic Uxml node N. Mediation Properties are held as strings.

The mediation message and properties constitute mediation task data.

As message data or any data including exchange variables and contributed data is based on generics, use XML API that is data model agnostic to process message data. For data manipulation you must use gXML. TIBCO gXML is an XML API that is based on generics and is data model agnostic.

This sample code shows processing message data:

```java
public class HelloWorldRT<I, U, N extends I, A extends I, S, T, X>
    extends Task<I, U, N, A, S, T, X> {

    public void init() throws TaskLifeCycleFault {  }
    public void destroy() throws TaskLifeCycleFault {  }
    public N execute(final N input, final Exchange<N> exchange)
        throws TaskFault {

        final GxProcessingContext<I, U, N, A, S, T, X> pcx =
            exchange.getXMLProcessingContext();

        final GxDocumentSerializerFactory<N, S> sf = new
            DocumentSerializerFactory<I, U, N, A, S, T, X>(pcx);

        // Configure for "pretty" printing.
        sf.setIndent(Boolean.TRUE);
        sf.setMethod(new QName("xml"));
        sf.setOmitXmlDeclaration(false);

        final StringWriter sw = new StringWriter();
        final GxDocumentSerializer<N> serializer =
            sf.newDocumentSerializer(sw);

        if(input != null){  serializer.serialize(input);  }else{
            serializer.serialize((exchange.getMessageData()));  }
```

TIBCO ActiveMatrix Service Grid Mediation Component Development
Logger logger = LoggerFactory.getLogger(HelloWorldRT.class);
logger.info(sw.toString());
return exchange.getMessageData();
}

Defining Model Attributes

A user-defined mediation task can support attributes that refer to a JDBC property. Developers of the mediation task can use a property to access JDBC connections using the mediation task API provided.

To create such a task, define a Property attribute type during the model-generation phase, using the model creation page in the mediation task model wizard:

![Model Creation Section](image)

This attribute type is projected by the mediation task’s user interface as a combination box that holds references to attributes defined at the mediation flow level.

Categorizing Custom Mediation Tasks

To create categories of custom mediation tasks, add the method `getPaletteGroup` method to the `MediationTaskUI` factory class as shown in this example:
public String getPaletteGroup(){
    return "Samples";
}

This example creates the category Samples.

**Thrown Faults**

The runtime class for a custom mediation task has an execute method that throws the fault TaskFault. A developer of a custom mediation task can throw this fault explicitly.

**Runtime Exceptions**

Table 15 explains handling runtime exceptions in a custom mediation task.

<table>
<thead>
<tr>
<th>Path</th>
<th>Exception Handling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Path control is transferred to the Catch Mediation Fault task in the fault path. A Send Fault task in that path sends a message to the service consumer.</td>
</tr>
<tr>
<td>Output</td>
<td>Path control is transferred to the Catch Mediation Fault task in the fault path. A Send Fault task in that path sends a message to the service consumer.</td>
</tr>
<tr>
<td>Fault</td>
<td>A fault message is sent to the service consumer.</td>
</tr>
</tbody>
</table>
Packaging Custom Mediation Tasks

To make your custom mediation tasks available in the Mediation Flow Editor, you must first install and deploy the plug-ins.

Installing Custom Mediation Tasks

To install the custom mediation tasks in TIBCO Business Studio:

1. Create a feature project.
   Specify the plug-in to package into the new feature.

2. Export the feature project.
   Make sure you check the check box for the Generate metadata repository option.

3. Install the feature using Help > Install New Software...
   Specify the location where you exported the feature project. Clear the check box for the Group items by category option which will then list the feature project.
   The custom task is ready for use and can be accessed from the palette.

Deploying Custom Mediation Tasks

After the file <runtime plug-in project name>\src\<runtime plug-in project name>\<MediationTaskName>rt.java is updated with the custom code, the deployable artifacts can be generated.

To create the deployable artifacts, complete the following steps:

1. Make sure the Target Platform points to ActiveMatrix Runtime.
   See Composite Development for information on switching the Target Platform.

2. Verify that the Model and Runtime plug-ins have no compilation errors.
3. In the Project Explorer pane, expand the `<runtime plug-in project name>.deploy.soap` project.

4. Expand the Composites folder.

5. Right-click `<runtime plug-in project name>.apt.composite`, and click Create DAA.

The Create Deployment Archive wizard is invoked. Refer to Composite Development for more information on using this wizard. Deploy the DAA that packages the custom mediation task Runtime plug-ins before deploying the mediation application that uses the custom task.

Refer to Administration for information on uploading and deploying the deployment application archive (.daa).

**Testing Custom Mediation Tasks**

To test the custom mediation task in RAD by creating a Run As/Debug As configuration:

1. Add one of the following to the Functions list along with the main composite:
   - A composite generated by the Custom Mediation Task wizards to the list
   - A DAA created from the composite

   Make sure that the composite or DAA that holds Custom Mediation Task is at the top of the list of Composite/DAA(s), before the SOA DAA/Composite.

2. Select Apply and Run/Debug.
Chapter 12  Reference

This chapter describes the configuration tabs for tasks and resources used in mediation flows.

Topics

- Catch Fault, page 176
- End Mediation, page 178
- Generate Reply, page 180
- Handle Reply, page 181
- Invoke Operation, page 182
- Log, page 184
- Parse XML, page 188
- Query Database, page 191
- Render XML, page 195
- Route Task, page 197
- Send Fault, page 201
- Set Context, page 202
- Set Dynamic Reference, page 204
- Set Exchange Variable, page 208
- Throw Fault, page 210
- Transform, page 211
- Validate XML, page 214
- XPath Route, page 218
Catch Fault

Task

The Catch Fault task specifies the faults that you want to catch from a target operation or a mediation flow.

Catch Fault tasks appear automatically in Fault paths. Catch Fault tasks do not appear in the palette, and you cannot add these tasks manually.

See Chapter 10, Fault Processing in a Mediation Flow, on page 141 for more information about fault processing.

General Tab

Use the General tab to select or clear specific faults to catch and handle. Selecting specific faults to catch creates a sub-path for each selected fault so that you can specify processing to perform for that fault before the fault is returned to the original environment.

Select the box in the Catch and Handle column for the fault you want to catch.

The Fault column provides a number of fault types:

<table>
<thead>
<tr>
<th>Item in Fault Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Declared Fault Message</td>
<td>The target operation, the Throw Fault task, and the Invoke Operation task can throw a declared fault message. The content and structure of the message varies, depending upon its declaration in the WSDL file.</td>
</tr>
<tr>
<td>UndeclaredFault</td>
<td>An undeclared fault that occurs while invoking the target operation returns this fault message.</td>
</tr>
<tr>
<td>MediationTaskFault</td>
<td>A declared fault that is thrown by one of the tasks in the mediation operation.</td>
</tr>
<tr>
<td>MediationTaskRuntimeFault</td>
<td>An undeclared fault that is thrown by one of the tasks in the mediation operation.</td>
</tr>
<tr>
<td>TimeoutFault</td>
<td>The TimeoutFault is returned when the invoked operation does not return in a specified time. The timeout value is configurable in the composite application.</td>
</tr>
<tr>
<td>Item in Fault Column</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Catch All</td>
<td>This item is always present and is selected when one or more other faults in the list are cleared. This item corresponds to the path for any faults that are not explicitly handled by other fault paths.</td>
</tr>
</tbody>
</table>
End Mediation

Task

The End Mediation task ends a one-way (in-only) or a Request-Response (in-out) message exchange pattern operation. One-way operations provide a way for service consumers to initiate operations for which they won’t receive a response—the End Mediation task is an orderly way to end the mediation execution. For example, you can log the operation’s input data using the Log task and then terminate the input path of the mediation operation with an End task.

The End Mediation task can be also configured for both in-only operation and in-out operation to signal the framework to redeliver the request message or stop re-delivery of the request message.

The mediation input path of a one-way message exchange pattern operation can contain other mediation tasks before terminating with the End Mediation task. However, if any of the other tasks in the mediation input path produces a fault at run-time, this will terminate the execution of the mediation input path and transfer control to the mediation fault path. No reply is sent to the consumer, because the fault path also terminates with an End Mediation task.

Using End Mediation from a one-way operation to a request-response target operation

You can mediate a one-way (in-only) message exchange pattern operation to a request-response (in-out) target operation. Although the mediation input path operation in this case is similar to that of a mediation flow containing a one-way operation to a request-response target operation, the behavior in the output and fault paths are different.

When mediating a one-way operation to a request-response operation, the target operation can either return a reply or throw a fault; Mediation Flow automatically terminates both with an End Mediation task:

- If the target operation returns a reply, the output path is executed and the path is terminated by the End Mediation task without sending a response to the requestor.
- If the target operation returns a fault, the target fault path is executed and the path is terminated by the End Mediation task without sending a fault to the requestor.

Intents and Policies

For a in-only message exchange pattern operation, if the end task is configured with either a Redeliver Message or Stop redeliver message option, an intent type of either At Least Once or One Way Transaction has to be defined for the mediation interface.
For a in-out message exchange pattern operation, if the end task is configured with either a Redeliver Message or Stop Redeliver Message option, an intent type of At Least Once has to be defined for the mediation interface.

When mediating one-way operations to request-response target operations, it is good practice to set a Log task to capture the response message on the output and fault paths, before the path execution stops at the End Mediation task.

**General Tab**

Use the General tab to specify a name and description for the task, and to specify how the End Mediation task is to exit.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Assign a name to the task, to identify the task in the mediation flow. This name appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.</td>
</tr>
<tr>
<td>Description</td>
<td>Describe the task briefly. This description appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.</td>
</tr>
<tr>
<td>End Type</td>
<td>Specifies how the End Mediation task exits:</td>
</tr>
<tr>
<td>Normal</td>
<td>stops the mediation flow immediately.</td>
</tr>
<tr>
<td>Redeliver Message</td>
<td>redelivers the message that initiated the mediation flow, re-executing the entire mediation flow.</td>
</tr>
<tr>
<td>Stop-Redeliver Message</td>
<td>stops the redelivery of messages.</td>
</tr>
<tr>
<td>Signal an Exception</td>
<td>generates an exception without enforcing an intent type of either At Least Once or One Way Transaction on the mediation flow and component.</td>
</tr>
</tbody>
</table>
Generate Reply

Task

The Generate Reply task terminates an input path and passes control to the Handle Reply task on the output path.

The Generate Reply task is used to create a reply to a mediation operation without passing the flow of control on to a target operation.

See Chapter 9, Replying to Messages, on page 135 for more information about the Generate Reply task.

General Tab

Use the General tab to specify a name and description for the task. This tab is useful for providing documentation for tasks in your mediation flows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Assign a name to the task, to identify the task in the mediation flow. This name appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.</td>
</tr>
<tr>
<td>Description</td>
<td>Describe the task briefly. This description appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.</td>
</tr>
</tbody>
</table>

Input Tab

The Input tab is a mapping panel that you use to map the mediation exchange to the output message of the operation. See Chapter 6, Transforming Data in a Mediation Exchange, on page 75 for more information about using a mapping panel.
Handle Reply

Task

The Handle Reply is the start of the mediation reply path for handling reply messages created by any Generate Reply task on the input path. A Handle Reply task appears automatically in the output path when a Generate Reply task is placed on the input path. Handle Reply tasks do not appear in the palette, and you cannot add these tasks manually.

See Chapter 9, Replying to Messages, on page 135 for more information about the Handle Reply task.

General Tab

Use the General tab to specify a name and description for the task. This tab is useful for providing documentation for tasks in your mediation flows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Assign a name to the task, to identify the task in the mediation flow. This name appears in the tooltip that opens when you hovers the cursor over the task icon in the mediation flow.</td>
</tr>
<tr>
<td>Description</td>
<td>Describe the task briefly. This description appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.</td>
</tr>
</tbody>
</table>
Invoke Operation

Task

The Invoke Operation task enables you to invoke an operation of an interface in the target interface list during processing of an input, output, or fault path. The operation can be one-way or request-reply. If the operation is request-reply, the reply message is stored in the mediation exchange for use by subsequent tasks in the mediation path.

See Chapter 3, Invoking an Operation, on page 49 for more information about the Invoke Operation task.

General Tab

Use the General tab to specify a name and description for the task. This tab is useful for providing documentation for tasks in your mediation flows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Assign a name to the task, to identify the task in the mediation flow. This name appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.</td>
</tr>
<tr>
<td>Description</td>
<td>Describe the task briefly. This description appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.</td>
</tr>
<tr>
<td>Target Operation</td>
<td>The operation to invoke. The drop-down list is populated with all operations from the interfaces in the target interface list of the mediation flow.</td>
</tr>
</tbody>
</table>
Input Tab

The Input tab is a mapping panel for mapping data from the mediation exchange to the input fields of this task. See Chapter 6, Transforming Data in a Mediation Exchange, on page 75 for more information about using a mapping panel.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task Input</td>
<td>A complex element containing the input message for the invoked operation. The structure of the sub-elements depends on the structure of the input message for the operation. You can also can input the required value for fields directly into the input schema.</td>
</tr>
</tbody>
</table>

Output Tab

The Output tab contains a static tree representation of the reply message schema for the invoked operation. Subsequent tasks in the mediation flow will have access to the reply message. The reply message is stored in an element within the mediation exchange whose root is named the same as the Invoke Operation task name specified on the General tab.

Subsequent tasks also have access to the message context properties in the reply message (see Working with Message Context Properties on page 30).

If the message exchange pattern for the operation is one-way, the output is null.
Log

Task

The Log task sends information from the mediation flow context to the log. The Log task can be placed on an input, output, or fault path.

For more information about the Log task, see Chapter 4, Logging Mediation Exchange Information, on page 53. For information about configuring the log, see Configuring Mediation Appenders and Loggers on page 58.

General Tab

Use the General tab to specify a name and description for the task. This tab is useful for providing documentation for tasks in your mediation flows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Assign a name to the task, to identify the task in the mediation flow. This name appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.</td>
</tr>
<tr>
<td>Description</td>
<td>Describe the task briefly. This description appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.</td>
</tr>
<tr>
<td>Log Role</td>
<td>Select the role for log messages—INFO, WARN, DEBUG, or ERROR. Each is handled separately, and each has its own log.</td>
</tr>
</tbody>
</table>

Use Transform Data

Select this option to display the Log tab as a mapping panel that shows the schema with the elements message, messageID, and role, so that you can build custom log messages. See Mapping Information for Custom Log Messages on page 186.

If this option is not set, the Log tab displays information from the mediation exchange for you to specify which information to send to the log file. See Specifying Information for Standard Log Messages on page 185.

The Use Transform Data option is cleared by default.
Log Tab

The appearance of the Log tab depends on whether the option **Use Transform Data** is selected on the General tab:

- If **Use Transform Data** is clear (the default), the Log tab shows top-level message information, from which you choose the information to log. See **Specifying Information for Standard Log Messages** on page 185.

- If **Use Transform Data** is selected, the Log tab appears as a mapping panel so that you can build custom log messages. See **Mapping Information for Custom Log Messages** on page 186.

Specifying Information for Standard Log Messages

If the **Use Transform Data** option is not selected in the General tab, you use the Log tab to specify what top-level information from the mediation exchange to send to the log file:

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log All Items</td>
<td>Selects all sub-items on this tab and sends all information in the mediation flow context to the log.</td>
</tr>
<tr>
<td>Mediation Flow Properties</td>
<td>The properties defined for the mediation flow. These properties can be defined on the Properties tab of the mediation flow. You can select the parent item to send all mediation flow properties to the log, or you can select individual properties to send the properties to the log.</td>
</tr>
<tr>
<td>Mediation Flow Context</td>
<td>Logs message context such as component and mediation flow name information, if the Mediation Flow Context option is set on the Advanced tab of the mediation operation Properties view. See Working with Message Context Properties on page 30 for information about the Mediation Flow Context option.</td>
</tr>
<tr>
<td>Message Context</td>
<td>Logs all message context information. The message context includes information about the transport used for the message and the security context for the message. You can optionally select either the transport or security information if you do not want the entire message context sent to the log.</td>
</tr>
</tbody>
</table>
Mapping Information for Custom Log Messages

If the **Use Transform Data** option is selected on the **General** tab, the Log tab is a mapping panel, where you can map mediation information to build custom log tasks. See Chapter 6, Transforming Data in a Mediation Exchange, on page 75 for more information about using a mapping panel.

<table>
<thead>
<tr>
<th>Item</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message Data</td>
<td>The content of the message.</td>
</tr>
<tr>
<td>Contributed Data</td>
<td>Some mediation tasks, such as Transform or custom tasks, can contribute additional data items to the mediation exchange. Each contributed data item is named for the task that contributes the data. You can send any contributed data item to the log.</td>
</tr>
<tr>
<td>Exchange Variable</td>
<td>You can send exchange variable information to the log if you have specified an exchange variable on the mediation operation, and have set it using the Set Exchange Variable task.</td>
</tr>
</tbody>
</table>

### Field Description

- **message**: Specify the data from the mediation exchange to log.
  
  You can log any data available in the mapper—the *message* element allows logging of a simple message, and also allows mapping XML documents in a serialized text form.

  If the **Mediation Flow Context** option is set on the **Advanced** tab of the mediation operation’s Properties view, you can map message context information to the *message* element. See Working with Message Context Properties on page 30 for information about the **Mediation Flow Context** option.
messageID | Optionally specify a message ID value to be included as part of the message that is being logged. The message ID consists of two elements, name and code. The name element is a string type and the code element is integer type.

At run-time, the value in the name element and the value in the code element are combined to form a message ID that has the syntax `name-code`. For example, if the name element contains the value `Mail` and the code element contains the value `1000` then the message ID will be `Mail-1000`.

However, if you only provide the value for the name element, a default value of `0` will be used for the code element. Similarly, if you only provide the value for code element, the default value for the name element will be `AMSB.LogTask`.

role | Optionally specify a logging-level role for run-time.

Values can be `info`, `warn`, `debug`, or `error`. Values are not case-sensitive.

If you map to this role, the value you give its property overrides the Log Role setting in the General tab.
**Parse XML**

*Task*

The Parse XML task is used when you have an XML document stored in a string or binary field. This task produces a tree representation of the XML that can be used by subsequent tasks in the mediation flow. This task can be paired with the Render XML task to convert the parsed XML back into a string or binary field for transmission within a message. XML documents are sometimes stored in string or binary fields to improve the performance of message transmission or for other reasons. You may want to view or manipulate the data within the document then replace the document in the message before transmission to a target operation or mediation operation. Also, the target or mediation operation in your mediation flow may expect to receive all or a subset of the fields within the document.

To parse an XML document, you must provide the schema definition for the data. The schema definition must be stored in an XSD within your project. You can use an existing XSD, create an XSD with the XSD editor within TIBCO Business Studio, or you can use your own XSD editor plug-in. See the Eclipse XSD Developer Guide for more information about the XSD editor within TIBCO Business Studio.

The output of the Parse XML task is placed into the contributed data portion of the mediation exchange. An element with the same name as the Parse XML task is placed into the mediation exchange. The XSD specified in the **Output Schema** field determines the structure of the element.

The Parse XML task can be placed on an input, output, or fault path.

**General Tab**

Use the **General** tab to specify a name, description, and output schema for the task. If the XML document is in binary format instead of text format, you can specify that on the General Tab.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Assign a name to the task, to identify the task in the mediation flow. This name appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow. This is also the name of the element in the mediation exchange that stores the output of this task.</td>
</tr>
</tbody>
</table>
### Input Tab

The Input tab is a mapping panel for mapping data from the mediation exchange to the input fields of this task. See Chapter 6, Transforming Data in a Mediation Exchange, on page 75 for more information about using the mapping panel.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TaskInput</td>
<td>A complex element to hold the input for this task. The sub-element of this element is the XML document that you want to parse. The Binary Input field on the General tab controls which of the following elements appear.</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>ParseXmlStringInput</td>
<td>Appears when the Binary Input field on the General tab is not selected. Map this element to a string element in the Mediation Exchange that holds the XML document you want to parse.</td>
</tr>
<tr>
<td>ParseXmlBinaryInput</td>
<td>Appears when the Binary Input field on the General tab is selected. Map this element to a binary element in the Mediation Exchange that holds the XML document you want to parse.</td>
</tr>
</tbody>
</table>

**Output Tab**

The **Output** tab is a read-only display of the output schema for this task.
Query Database

The Query Database task is used to construct a SQL SELECT statement query to a database. This task is useful for performing basic queries for looking up information stored in a database table that will be used in the mediation flow.

General Tab

Use the General tab to specify a name and description for the task. This tab is useful for providing documentation for tasks in your mediation flows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Assign a name to the task, to identify the task in the mediation flow. This name appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow. This is also the name of the element in the mediation exchange that stores the output of this task.</td>
</tr>
<tr>
<td>Description</td>
<td>Describe the task briefly. This description appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.</td>
</tr>
<tr>
<td>Shared Resource Property</td>
<td>Use the drop-down list on the field to select the available shared resource property. Shared resource properties define database connections that are used to perform the query. Shared resource properties are defined on the Shared Resource Properties tab of the mediation flow.</td>
</tr>
<tr>
<td>Max Row Count</td>
<td>The maximum number of rows to retrieve. The default value is 1. Specify a positive integer, or use the drop-down menu on the field to select Unlimited to return an unlimited number of rows.</td>
</tr>
<tr>
<td>Query Timeout (sec)</td>
<td>The timeout for the query.</td>
</tr>
</tbody>
</table>

Query Tab

You use the Query tab to define the SELECT statement for the query.
Click the connection button to test the connection and to verify the table and column data. Clicking this button opens a connection, if you have specified a JDBC property in the mediation flow Properties tab, and compares the table and column data with the metadata from the database. If the connection is not successful, an error notifies you of the reason.

Three lists enable you to select tables, input data, and output columns for use in the WHERE clause of your SELECT statement.

Input data is used in the WHERE clause of your SELECT statement. Use the add (+) and delete (x) buttons to the right of each list to add and delete items from each list:

- When a database connection is present and valid, the + buttons display information from the database for selecting tables and output columns.
- When no database connection is present, the + buttons allow you to add items to each list, but you must name each item and specify a type if necessary.

Clicking the + and x buttons on the Input table attempt an automatic update of the WHERE condition. If you have modified the WHERE condition, the delete might not update it and you must fix it manually.

Use the Where Condition field on the Query tab to edit the WHERE clause of the query. You can add an input variable to a condition by typing a question mark (?) in the condition. Each input variable appears in the mapper panel on the Input tab, and you can supply data from the mediation exchange for the input variable. For example, if you want to create a condition to look up a zip code supplied in the input message, you can add the condition `table.ZIP = ?`. When you add a question mark into the WHERE clause, an input variable appears in the Input Data list. Supply a name for the input variable, then data from the mediation exchange can be mapped to the input variable.

Table join conditions are never automatically added to the WHERE clause, so you must manually edit the WHERE clause to specify any join conditions for your query.

The SQL Statement field displays a read-only version of the query you have specified. Table 16 lists the supported SQL types and how they map to XML. Note that length parameters are stripped from the SQP Type, and only the base type is used in the mapping—for example, `char(12)` becomes `char`.

<table>
<thead>
<tr>
<th>SQL/92 Data Types</th>
<th>XML Type Equivalent</th>
</tr>
</thead>
<tbody>
<tr>
<td>TINYINT</td>
<td>short</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>short</td>
</tr>
</tbody>
</table>
Vendor-specific types are cast to string. You can enable the mapper to automatically recognize these types in one of these ways:

- Force vendor-specific types to a compatible XML type using the mapper cast.
- Override the type that is retrieved from the database for the column to a similar SQL/92 type.

Binary or other complex data types such as JAVA_OBJECT are not supported.
Input Tab

The **Input** tab is a mapping panel for mapping data from the mediation exchange to the input fields of this task. See Chapter 6, Transforming Data in a Mediation Exchange, on page 75 for more information about using a mapping panel.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>InputValues</td>
<td>A complex element to hold the input for this task. The sub-elements of this element are the input variables defined on the <strong>Query</strong> tab. Each input variable corresponds to a question mark (?) that appears in the WHERE clause of the query. Map values from the mediation exchange to fields in the input schema to supply values for the input variables of the query.</td>
</tr>
</tbody>
</table>

Output Tab

The **Output** tab is a read-only display of the output schema for this task. The output schema is determined by the output columns selected on the **Query** tab.

Test Tab

The **Test** tab is used to test the database query. You must have a valid JDBC template associated with the JDBC property used by this task. The JDBC resource template is used only in the design environment.

You can use a custom JDBC driver to test the database query. For information about configuring a custom JDBC driver, see *Composite Development*.

It is important for you to ensure that the JDBC resource template you use for testing in the design environment connects to a database that is similar to the database used when the project is put into production.
Render XML

Task

The Render XML task takes an XML tree for a specified schema and converts it to a string or binary element that contains the XML document. This task can be paired with the Parse XML task to convert the parsed XML back into a string or binary field for transmission within a message.

XML documents are sometimes stored in string or binary fields to improve the performance of message transmission or for other reasons. You may want to view or manipulate the data within the document then replace the document in the message before transmission to a target operation or mediation operation. Also, the target or mediation operation in your mediation flow may expect to receive all or a subset of the fields within the document.

To render an XML document, you must provide the schema definition for the data. The schema definition must be stored in an XSD within your project. You can use an existing XSD, create an XSD with the XSD editor within TIBCO Business Studio, or you can use your own XSD editor plug-in. See the Eclipse XSD Developer Guide for more information about the XSD editor within TIBCO Business Studio.

The output of the Render XML task is placed into the contributed data portion of the mediation exchange. An element with the same name as the Render XML task is placed into the mediation exchange. The contents of the element is either a string or binary element containing the XML document.

The Render XML task can be placed on an input, output, or fault path.

General Tab

Use the General tab to specify a name, description, and input schema for the task. If the XML document will be stored in binary format instead of text format, you can specify that on the General Tab.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Assign a name to the task, to identify the task in the mediation flow. This name appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow. This is also the name of the element in the mediation exchange that stores the output of this task.</td>
</tr>
</tbody>
</table>
Input Tab

The Input tab is a mapping panel for mapping data from the mediation exchange to the input fields of this task. See Chapter 6, Transforming Data in a Mediation Exchange, on page 75 for more information about using a mapping panel.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TaskInput</td>
<td>A complex element to hold the input for this task. The sub-element of this element is the schema specified in the Input Schema field on the General tab. Map values from the mediation exchange to fields in the input schema to create the XML document.</td>
</tr>
</tbody>
</table>

Output Tab

The Output tab is a read-only display of the output schema for this task.
Route Task

The Route task enables you to send messages to a specific destination based on conditions that you specify. Data from the mediation flow context, such as the security information or message body, can be used to specify the conditions of the route.

You can only introduce the route in the input path. The response (output or fault) always returns to the original requester—that is, to the same mediation operation.

Route tasks send each incoming message to a single destination based on which route case evaluates to true, or to a single destination designated as otherwise if none of the cases evaluate to true.

You can use multiple, nested Route tasks to send a single message to a target in several different ways, based on the routing cases, conditions, and variables you set for each task in the Decision tab.

You can configure multiple routes in an input flow, nesting them to any depth, and you can place mediation tasks on flow paths before or after any route task. This enables users to decide which tasks are executed in common and which are executed only for specific route cases.

See Chapter 5, Routing Messages in a Mediation Flow, on page 59 for more information about the Route task.

General Tab

Use the General tab to specify a name and description for the task. This tab is useful for providing documentation for tasks in your mediation flows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Assign a name to the task, to identify the task in the mediation flow. This name appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.</td>
</tr>
<tr>
<td>Description</td>
<td>Describe the task briefly. This description appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.</td>
</tr>
</tbody>
</table>
Decision Tab

Use the Decision tab to create routing cases, routing conditions, and variables to hold data that will be evaluated in the routing conditions. The Decision tab has a toolbar for adding and deleting cases and conditions:

<table>
<thead>
<tr>
<th>Toolbar Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="Add Case" /></td>
<td><strong>Add Case</strong> Adds a routing case to the table on this tab. A new case appears in the table on this tab with a default name, and the case is drawn in the mediation flow diagram. By default, new cases created with this icon point to an error icon until a Target Service/Operation is specified for the case.</td>
</tr>
<tr>
<td><img src="image" alt="Delete Case" /></td>
<td><strong>Delete Case</strong> Deletes the selected routing case.</td>
</tr>
<tr>
<td><img src="image" alt="Add Variable" /></td>
<td><strong>Add Variable</strong> Adds a variable to use in routing conditions. Clicking this icon opens a dialog that enables you to specify the name and data type of the variable. These datatypes are available:</td>
</tr>
<tr>
<td></td>
<td>• string</td>
</tr>
<tr>
<td></td>
<td>• integer</td>
</tr>
<tr>
<td></td>
<td>• boolean</td>
</tr>
<tr>
<td></td>
<td>• date</td>
</tr>
<tr>
<td></td>
<td>• time</td>
</tr>
<tr>
<td></td>
<td>• float</td>
</tr>
<tr>
<td></td>
<td>Each variable appears as a column between the Case column and the Target Service/Operation column.</td>
</tr>
<tr>
<td><img src="image" alt="Delete Variable" /></td>
<td><strong>Delete Variable</strong> Opens a dialog for you to select the variable to delete.</td>
</tr>
</tbody>
</table>

The Decision tab includes a table containing all of the routing cases:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case</td>
<td>Name of the routing case. Click in the cell to edit the name.</td>
</tr>
</tbody>
</table>
At the bottom of the **Decision** tab is the configuration for the Otherwise case for the route. The Otherwise case is taken when all other cases evaluate to false. You can specify the target operation for this case in the **Target Service/Operation** field.

Also at the bottom of the **Decision** tab is a drop-down list of choices for setting the case target to a specific type of mediation task. For example, selecting **Throw Fault** sets the target to a new **Throw Fault** task.

Targets you can specify are:

- Targeted operations that are not already targeted
- Generate Reply, **Throw Fault** for mediation tasks
- End Mediation for one-way (in-only) operations.

### Variable List

Variables created with the **Add Variable** icon appear as columns in this table. You must specify a comparison operator and a constant to compare for each variable. Click the cell to select the comparison operator from a drop-down list and edit the value of the constant in the text field.

You can specify comparison operators:

- \( = \) (equal)
- \( \neq \) (not equal)
- \(<\) (less than)
- \(\leq\) (less than or equal)
- \(>\) (greater than)
- \(\geq\) (greater than or equal)

All variable conditions that you specify for each case must evaluate to true for the case to evaluate to true.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable List</td>
<td>Variables created with the <strong>Add Variable</strong> icon appear as columns in this table. You must specify a comparison operator and a constant to compare for each variable. Click the cell to select the comparison operator from a drop-down list and edit the value of the constant in the text field. You can specify comparison operators:</td>
</tr>
<tr>
<td>Target Service/Operation</td>
<td>The name of the Target Service and target operation that is the destination for this case. If you drag the path for the case to a target operation in the mediation flow, this field is automatically set to the correct value. You can also click this field to either type or select the target operation.</td>
</tr>
</tbody>
</table>
- Route tasks and XPath Route tasks, which enables you to build nested routing structures.

Any change you make to a nested routing structure replaces the entire nested structure.

**Input Tab**

Use the **Input** tab to map data from the mediation exchange into the list of variables that you have created for the Route task. See Chapter 6, Transforming Data in a Mediation Exchange, page 75 for a complete description of how to perform mapping.
Send Fault

**Task**

The Send Fault task returns a fault message to the original process that invoked the mediation task.

Send Fault tasks appear automatically in Fault paths. Send Fault tasks do not appear in the palette, and you cannot add these tasks manually.

See Chapter 10, Fault Processing in a Mediation Flow, on page 141 for more information about fault processing.

**General Tab**

Use the **Fault** tab to specify the fault to send to the original environment:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Fault to Send</strong></td>
<td>A drop-down list of declared fault messages on the mediation operation. You can also choose to send the UndeclaredFault message.</td>
</tr>
</tbody>
</table>
Set Context

Task

The Set Context task provides a way to set the values for the message context properties of the target operation’s input message and the message context properties of the mediation operation’s output message. This allows the mediation path to set the message context data (such as HTTP header or JMS user properties) for the output message of the mediation operation and the input message of the target operation.

The schema that appears on the Set Context task is configured in the Properties view of the mediation operation or target operation. On the Advanced tab, you can set the field Message Context Properties (outbound) of the mediation operation, or the Message Context Properties (inbound) field of the target operation.

See Working with Message Context Properties on page 30 for more information.

General Tab

Use the General tab to specify a name and description for the task, and to identify the operation for which to set the context.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Assign a name to the task, to identify the task in the mediation flow. This name appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.</td>
</tr>
<tr>
<td>Description</td>
<td>Describe the task briefly. This description appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.</td>
</tr>
<tr>
<td>Operation</td>
<td>The operation for the Set Context task. The drop-down list is populated with all operations from the interfaces in the target interface list of the mediation flow. If the path is leading to the mediation Operation (for example, the Output path), you can select the mediation operation and set the context properties of the mediation output message. The Operation field identifies this case by identifying the interface (for example: [Mediation Interface]:HelloWorld/sayHello).</td>
</tr>
</tbody>
</table>
Input Tab

The **Input** tab is a mapping panel for mapping data from the mediation exchange to the input fields of this task. See Chapter 6, Transforming Data in a Mediation Exchange, on page 75 for more information about using a mapping panel.

The input context for the target operation appears in the right side of the mapper. The task input structure provides context properties:

- Of the operation’s outbound message, if a mediation operation is selected in the **General** tab.
- Of the operations’s inbound context message, if a target operation is selected in the **General** tab.
Set Dynamic Reference

Task

The Set Dynamic Reference task provides the values needed for resolving a service provider in a dynamic target interface. Each Set Dynamic Reference task sets the value of the service provider for the specified dynamic target interface—either the end of the mediation path points to a dynamic target interface, or an Invoke task invokes an operation on a dynamic target interface.

General Tab

Use the General tab to specify a name and description of the Set Dynamic Reference task, and to set the target interface and endpoint reference mechanism.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Assign a name to the task, to identify the task in the mediation flow. This name appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.</td>
</tr>
<tr>
<td>Description</td>
<td>Describe the task briefly. This description appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.</td>
</tr>
<tr>
<td>Dynamic Target Interface</td>
<td>The name of the dynamic target interface for which this task is supplying the service name and namespace. By default, this field is automatically set to the dynamic target interface at the other end of the path. If an Invoke task is on the path, this field may be set by default to the first dynamic target interface in the target interface list. You might need to set this field when the default choice does not match the dynamic target interface that you want to set. The name of the dynamic target interface for which this task is supplying the application and service name. By default, this field is set to the dynamic target interface at the other end of the path. If there is an Invoke task on the path, this field can be set by default to the first dynamic target interface in the target interface list. You might need to set this field when the default choice does not match the dynamic target interface that you want to set.</td>
</tr>
</tbody>
</table>
When the **Endpoint Reference Mechanism** field on the **General** tab is set to Service & Namespace, the input elements for this task are `serviceName` and `serviceNamespace`. Any value you specify for these input elements overrides the value specified on the **Pattern Map** tab of the specified dynamic target interface.

### Field Description

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Endpoint Reference Mechanism</strong></td>
<td>Select an option for the mechanism to use for setting the application and service name:</td>
</tr>
<tr>
<td></td>
<td>• <strong>Application &amp; Service Name</strong> Select to supply the application name and service name. This option requires two inputs for mapping on the <strong>Input</strong> tab—ApplicationName and ServiceName.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Application &amp; Service URI</strong> Select to supply the exact URI of the endpoint. This is useful if, for example, someone sends you the URI—you simply copy and paste it into the ApplicationServiceURI parameter on the <strong>Input</strong> tab.</td>
</tr>
<tr>
<td></td>
<td>• <strong>URI</strong> Select this option to specify a URI.</td>
</tr>
<tr>
<td></td>
<td>• <strong>Pattern Variables</strong> Select to use pattern variables from the dynamic target interface. This is useful if several operations in a dynamic target interface use a similar pattern for the application name and service name. You can specify the mapping once on the dynamic target interface and use variables to supply the variable portion. The variables you create on the dynamic target interface appear in the <strong>Input</strong> tab when this option is selected.</td>
</tr>
</tbody>
</table>

By default, the **Endpoint Mechanism** field is set to Application & Service Name when the dynamic target interface has no pattern variables.

If the dynamic target interface has pattern variables, the **Endpoint Mechanism** field is set to Pattern Variables by default.

### Input Tab

When the **Endpoint Reference Mechanism** field on the **General** tab is set to Service & Namespace, the input elements for this task are `serviceName` and `serviceNamespace`. Any value you specify for these input elements overrides the value specified on the **Pattern Map** tab of the specified dynamic target interface.
When the **Endpoint Reference Mechanism** field on the **General** tab is set to Pattern Variables, then the pattern variables defined on the specified dynamic target interface are the input elements. This enables you to specify simple mappings of data from the mediation exchange to the variable values. The variable values are then passed to the mapping supplied on the **Pattern Map** tab of the dynamic target interface.

Use any data available in the mediation exchange on the left side of the mapper panel to provide data to the input values.

The content of the **Input** tab depends on which Endpoint Reference Mechanism you select on the **General** tab—Application & Service Name, Application & Service URI, or Pattern Variables:

- **Application & Service Name** When the **Endpoint Reference Mechanism** field on the **General** tab is set to Application & Service Name, the input elements for the Set Dynamic Reference task are **ApplicationName** and **ServiceName**:
  - **ApplicationName** refers to the application name provided during deployment of a composite, to uniquely identify an instance of an application template.
  - **ServiceName** is the name of the composite service that is contained in the target composite.

- **Application & Service URI** When the **Endpoint Reference Mechanism** field on the **General** tab is set to Application & Service URI, the input element for the Set Dynamic Reference task is **ApplicationServiceURI**. The data type of the **ApplicationServiceURI** input field is a URI of the format `urn:amx:EnvironmentName/ApplicationName#service(ServiceName)`

  The variables **EnvironmentName**, **ApplicationName**, and **ServiceName** refer to the environment and service that are being invoked:
  - **EnvironmentName** is the name of the ActiveMatrix environment that contains the target service.
  - **ApplicationName** refers to the application name that is provided during deployment of a composite, to uniquely identify an instance of an application template.
  - **ServiceName** is the name of the composite service that is contained in the target composite.

- **URI** When the **Endpoint Reference Mechanism** field on the **General** tab is set to URI, the input elements for the Set Dynamic Reference task are **URI**:
  - **SOAP over HTTP**
    
    $http://<HostName>:<PortNumber>/<Path>$
<PortNumber> and <Path> are optional elements.

- SOAP over JMS: Specify the queue as
  
  jms:queue:<QueueName>

- ActiveMatrix Service Virtualization
  
  urn:amx:<EnvironmentName>/<ApplicationName>#service(<PromotedServiceName>)

- **Pattern Variables** When the **Endpoint Reference Mechanism** field on the **General** tab is set to Pattern Variables, the pattern variables that are defined on the specified dynamic target interface are the input elements.

  This enables you to specify simple mappings of data from the mediation exchange to the variable values. The variable values are then passed to the mapping supplied on the **Pattern Map** tab of the dynamic target interface. You can use any data available in the mediation exchange on the left side of the mapper panel to provide data to the input values.
Set Exchange Variable

Task

The Set Exchange Variable task sets the value of the exchange variable mediation exchange.

The Set Exchange Variable task can be placed on an input, output, or fault path.

The Set Exchange Variable task sets the value of the entire exchange variable—if you need to set several attributes, set them all at once, using one Set Exchange Variable task.

See Working with Exchange Variables on page 37 for a description of how to define exchange variables for mediation operations.

General Tab

Use the General tab to specify a name and description for the task. This tab is useful for providing documentation for tasks in your mediation flows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Assign a name to the task, to identify the task in the mediation flow. This name appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.</td>
</tr>
<tr>
<td>Description</td>
<td>Describe the task briefly. This description appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.</td>
</tr>
</tbody>
</table>
**Input Tab**

The **Input** tab contains a mapping panel for mapping data from the mediation exchange to the input fields of this task. See *Chapter 6, Transforming Data in a Mediation Exchange, on page 75* for more information about using a mapping panel.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ExchangeVariable</td>
<td>The exchange variable element defined in the mediation operation <strong>General</strong> tab appears on the right side of the mapper.</td>
</tr>
<tr>
<td></td>
<td>To set the exchange variable, map values from the mediation exchange to the exchange variable element.</td>
</tr>
</tbody>
</table>
Throw Fault

Task

The Throw Fault task enables you to explicitly throw a fault in a mediation flow.

This task can be placed only on the input path.

The Throw Fault task is useful in these situations:

- You want to deprecate a mediation operation, and send a fault to all clients that request that operation.

- You want to specify routing cases where a fault should be sent. For example, if a loan processing application cannot process loans over $5,000,000, then you would configure a routing case for the loan request operation to examine the loan amount and place a Throw Fault task on the sub-path when the loan amount is over $5,000,000.

See Chapter 10, Fault Processing in a Mediation Flow, on page 141 for more information about fault processing.

General Tab

Use the General tab to select the fault to throw. You can choose to throw the MediationTaskFault message, or you can click Browse to open a dialog of service descriptors. You can choose from the list of messages in the selected service descriptors to send a specific message when a fault is thrown. When a message in a service descriptor is selected, the WSDL Location field appears. You can click the field label link to view the service descriptor in the WSDL editor.
**Transform**

*Task*

The Transform task is used to manipulate the data available in a mediation flow so that the expected input, output, or fault message can be created.

Transform tasks can be placed on input, output, or fault paths.

See Chapter 6, Transforming Data in a Mediation Exchange, on page 75 for more information about Transform tasks.

**General Tab**

Use the General tab to specify a name and description for the task. This tab is useful for providing documentation for tasks in your mediation flows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Name</strong></td>
<td>Assign a name to the task, to identify the task in the mediation flow. This name appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.</td>
</tr>
<tr>
<td><strong>Description</strong></td>
<td>Describe the task briefly. This description appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.</td>
</tr>
<tr>
<td><strong>Contribute Output to Mediation Exchange</strong></td>
<td>When not selected, this option signifies that the output of the Transform task should change the message data. When this option is selected, the message data is left unchanged, and the output of this task is added as another data item within the mediation exchange. The data contributed by this task is available to subsequent mediation tasks along the same path. If you select Use External Stylesheet on this tab, the Contribute Output to Mediation Exchange option is automatically selected and cannot be cleared.</td>
</tr>
<tr>
<td><strong>Use External Stylesheet</strong></td>
<td>Select to use an external stylesheet for data transformation. This enables you to specify the transformation mapping in your workspace, outside the mediation flow.</td>
</tr>
</tbody>
</table>
## Input and Output Style

Specify how the XML will appear:
- **Text** Specified with a string.
- **Binary** Specified with a binary value.
- **Tree** Specified with an *any* element, so that you can transform data already in an XML document.

## Stylesheet Reference Type

Select the type of reference for the Transform task:
- A static reference enables you to select a single (static) stylesheet from a folder that is in your project.
- A dynamic reference enables you to select a set of stylesheets from a folder in the project. At run-time one of the stylesheets in the list will be used dynamically, based on the value provided for the stylesheetURI element that is in the Input tab of the mediation task.

For example, if the folder specified for the dynamic reference is `MySOAProject/Service Descriptors` and the stylesheet is in the folder `MySOAProject/Service Descriptors/folder1/sample.xsl`, the value that must be provided for the stylesheetURI element must be `folder1/sample.xsl`.

When you specify a folder for dynamic reference, ActiveMatrix recursively includes the stylesheets under this folder and its sub-folders.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input and Output Style</td>
<td>Appears only if you select <strong>Use External Stylesheet</strong>. Specify how the XML will appear:</td>
</tr>
<tr>
<td>Stylesheet Reference Type</td>
<td>Appears only if you select <strong>Use External Stylesheet</strong>. Select the type of reference for the Transform task:</td>
</tr>
<tr>
<td>Static Style Sheet Reference</td>
<td>Appears if you select a static stylesheet reference type. Click <strong>Browse</strong> to select a single (static) stylesheet that is in your workspace.</td>
</tr>
<tr>
<td>Dynamic Stylesheet Folder</td>
<td>Appears if you selected a dynamic stylesheet reference type. Chose a value available in the drop-down list. At run-time, one of the style sheets in the list will be used dynamically, based on the input to the mediation task.</td>
</tr>
</tbody>
</table>
Input Tab

Use the **Input** tab to map data from the mediation exchange into the expected message schema.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>xmlString</td>
<td>Specify an XML document serialized as a string.</td>
</tr>
<tr>
<td>xmlBinary</td>
<td>Specify an XML document serialized in Base64Binary format.</td>
</tr>
<tr>
<td>xmlTree</td>
<td>Specify an XML document.</td>
</tr>
<tr>
<td>stylesheetURI</td>
<td>Specify the schema URI, so that ActiveMatrix can locate it at run time and use it for the transformation.</td>
</tr>
<tr>
<td>parameter</td>
<td>A stylesheet can expect zero, one or more parameter(s) for its execution at runtime:</td>
</tr>
<tr>
<td></td>
<td><strong>Parameter Name</strong>—Name of the parameter the stylesheet expects.</td>
</tr>
<tr>
<td></td>
<td><strong>Parameter Value</strong>—Value of the parameter.</td>
</tr>
</tbody>
</table>

See Chapter 6, *Transforming Data in a Mediation Exchange*, page 75 for a description of how to use a mapping panel.

Output Tab

The **Output** tab shows a tree representation of the Transform task output. Depending on the input style chosen, the output can be:

- xmlString—XML document serialized as string
- xmlBinary—XML document serialized in Base64Binary format
- xmlTree—XML document
Validate XML

Task

You use the Validate XML task to validate message data, a WSDL message, XML text, binary, or XML tree formats against a schema. The output of the Validate XML task is contributed to the mediation exchange, and can be used by downstream tasks.

Validate XML processes an XML document against an XML schema, to report any errors found. It does not produce a parsed tree.

You choose the schema against which validation is to be performed by first specifying its reference type in the General tab of the Validate XML task:

- A static reference enables you to select a single (static) schema from a folder that is in your project.
- A dynamic reference enables you to select a set of schemas from a folder that is in your project. At run-time one of the schemas in the list will be used dynamically for validation, based on the input to the mediation task. When you specify a folder for dynamic reference, ActiveMatrix recursively includes the schemas under this folder and its sub-folders.

The schema for a reference must be located in the same project as the mediation flow that uses it.

General Tab

On the General tab you specify a name and description for the task, and specify the type of schema to be used during verification.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Assign a name to the task, to identify the task in the mediation flow. This name appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.</td>
</tr>
<tr>
<td>Description</td>
<td>Describe the task briefly. This description appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>Throw Fault on Validation Error</strong></td>
<td>When this option is set, an error in the Validate XML task results in a MediationTaskFault.</td>
</tr>
<tr>
<td></td>
<td>When this check box is not selected, an error in the Validate XML task produces output that contains two fields:</td>
</tr>
<tr>
<td></td>
<td><strong>valid</strong> has two values: <em>true</em> if the XML is valid; <em>false</em> if the XML is invalid.</td>
</tr>
<tr>
<td></td>
<td><strong>error</strong> appears if the XML validation fails, and contains an errorCode and an errorMessage. These codes follow the W3C specification for XML schema.</td>
</tr>
</tbody>
</table>

**Input**

Select the input type:

- **MessageData**—Validates the mediation or target operation’s input data. This option reads the input message itself, so no mapping is required. MessageData is the default input type.
- **WSDL Message**—Validates the input or output of any operation in the WSDL.
  
When you select the **WSDL Message** option, the **Schema Source** field opens. Select the mediation or target option, and the message type (input or output).

  - **Text**
  - **Binary**
  - **XML Tree**

**Schema Reference Type**

When you select an input type of Text, Binary, or XML Tree, you can choose a static or dynamic reference type:

- A static reference enables you to select a single (static) schema from a folder that is in your project.
- A dynamic reference enables you to select a set of schemas from a folder that is in your project. At run-time one of the schemas in the list will be used dynamically for validation, based on the input to the mediation task. When you specify a folder for dynamic reference, the ActiveMatrix recursively includes the schemas under this folder and its sub-folders.
Input Tab

The content of the **Input** tab depends on the Input type you selected on the **General** tab:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MessageData</td>
<td>No mapping is required in the <strong>Input</strong> tab.</td>
</tr>
<tr>
<td>WSDL Message</td>
<td>Displays a message tree corresponding to the operation and message selected in the <strong>General</strong> tab.</td>
</tr>
<tr>
<td>Text</td>
<td>Specify the <code>xmlString</code> input to validate.</td>
</tr>
<tr>
<td>Binary</td>
<td>Specify the <code>xmlBinary</code> input to validate.</td>
</tr>
<tr>
<td>Tree</td>
<td>In the <code>xmlTree</code> node, specify any element to validate.</td>
</tr>
</tbody>
</table>

Output Tab

The **Output** tab of the Validate XML task shows the results of the validation, indicating whether the incoming XML is valid or invalid, after being verified against the specified schema.
If validation fails, an error description identifies the cause of the failure. You can log this error description for design-time troubleshooting.

If the **Throw Fault** field is selected, no output is produced by this task. The **Output** tab shows a tree with the message No Output Configured.
XPath Route

Task

The XPath Route task enables you to send messages to a specific destination based on conditions that you specify. Data from the mediation exchange, such as the security information or message body, can be used to specify the conditions of the route.

XPath Route tasks can only be placed on input paths, but specifying an XPath Route task on the input path automatically creates the correct output and fault paths.

See Chapter 5, Routing Messages in a Mediation Flow, on page 59 for more information about the XPath Route task.

General Tab

Use the General tab to specify a name and description for the task. This tab is useful for providing documentation for tasks in your mediation flows:

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Assign a name to the task, to identify the task in the mediation flow. This name appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.</td>
</tr>
<tr>
<td>Description</td>
<td>Describe the task briefly. This description appears in the tooltip that opens when you hover the cursor over the task icon in the mediation flow.</td>
</tr>
</tbody>
</table>

Decision Tab

Use the Decision tab to create routing cases, routing conditions, and variables to hold data that will be evaluated in the routing conditions. The Decision tab includes a toolbar for adding and deleting cases and conditions:

<table>
<thead>
<tr>
<th>Toolbar Icon</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add Case</td>
<td>Adds a routing case to the table on this tab. A new case appears in the table on this tab with a default name, and the case is drawn in the mediation flow diagram. By default, new cases created with this icon point to an error icon until a Target Service/Operation is specified for the case.</td>
</tr>
</tbody>
</table>
The Decision tab includes a table containing all of the routing cases:

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Case</td>
<td>Name of the routing case. Click the cell to edit the name.</td>
</tr>
</tbody>
</table>
At the bottom of the table is the configuration for the Otherwise case for the route. The Otherwise case is taken when all other cases evaluate to false. Use the Target Service Operation field to specify the target operation to perform for this case.

You can use the XPath editor window at the bottom of the Decision tab to edit the XPath expressions for each routing condition.

<table>
<thead>
<tr>
<th>Column</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Variable Names</td>
<td>The name of each variable that you create appears in at the top of the middle column of the table. The middle column is used to specify the XPath expression for the routing condition for each case. Your expressions are not limited to simple comparisons, and you do not need to use any of the variables you have defined in the expressions. You must type the XPath expression in the condition field next to each routing case, or select the field and use the Xpath Editor field at the bottom of the tab to edit the expression. Variables are referenced in the XPath expressions for each routing case by their names. Unlike XPath expressions in the Transform task, you do not need to use a dollar sign to specify the root of the path to the variable. For example, the expression to determine if the city variable is equal to &quot;Palo Alto&quot; would be: city = &quot;Palo Alto&quot; The Transform task has a graphical XPath editor that you can use as a reference for creating XPath functions for the route task. See Using XPath on page 107 and Data/Function Tabs on page 91 for more information about XPath.</td>
</tr>
<tr>
<td>Target Service/Operation</td>
<td>The name of the Target Service and target operation that is the destination for this case. If you drag the path for the case to a target operation in the mediation flow, this field is set automatically to the correct value. You can also click this field to either type or select the target operation.</td>
</tr>
</tbody>
</table>
Input Tab

Use the Input tab to map data from the mediation exchange into the list of variables that you have created for the XPath Route task. See Chapter 6, Transforming Data in a Mediation Exchange, page 75 for a description of how to use a mapping panel.
Appendix A  TIBCO AutoMediate Command-Line Tool

Topics

- Overview, page 224
- How the AutoMediate Command Line Tool Creates a SOA Project, page 225
- Running the AutoMediate Command Line Tool, page 226
- Deploying the Generated Deployment Application Archive, page 227
- AutoMediate Command Syntax and Options, page 228
- AutoMediate ANT Command Syntax and Options, page 232
Overview

TIBCO AutoMediate Command Line is an independent command-line tool that enables you to quickly on-ramp a large number of existing services, without having to use TIBCO Business Studio, the Mediation Flow Editor, or the Composite Editor to build the necessary design-time components.

The AutoMediate Command Line tool uses existing services as input, specified in a concrete WSDL, to create a fully functional composite application with pass-through mediation capabilities. The tool generates a deployment artifact archive that you can deploy into the runtime environment as the first step in establishing an Enterprise Service Bus (ESB).

By establishing an ESB pattern using the AutoMediate Command Line tool and ActiveMatrix, you virtualize existing provided or consumed services so that they become location-transparent and more adaptable to change:

- Virtualizing provided services hides the location of service providers, helping to avoid interrupting clients that are using the services.
- Virtualizing consumed services hides the details of how the services are provided, helping to avoid interrupting logic that depends on the services.

The AutoMediate Command Line tool builds SOA projects that contain these components:

- A pass-through mediation flow.
- A composite that contains the mediation component wired to services and references, depending on the number of ports specified in the concrete WSDL.
- The deployment artifact file for the generated composite application.

You can save these components in your source control system, then add mediation capabilities—for example routing, transformation, and validation—whenever changes are necessary.
How the AutoMediate Command Line Tool Creates a SOA Project

This diagram illustrates how the AutoMediate Command Line tool creates a SOA project from a concrete WSDL with HTTP service bindings:

In this diagram:

- The Service is a SOAP service. The host can be configured during deployment using HTTP connector name generated for the service.
- The AutoMediate Command Line tool extracts binding information from the WSDL and uses it in the Reference. The tool also specifies the binding information as a substitution variable that can be replaced at run-time, if the Reference has changed its location.
Running the AutoMediate Command Line Tool

To run the AutoMediate Command Line tool:

1. The AutoMediate Command Line tool is available in the
   `<TIBCO_HOME>/amx_it_mediation/<version>/bin` location. Navigate to
   this folder and open a command window.

2. At the prompt, type the `AutoMediate` command, specifying the concrete
   WSDL and any options. (See AutoMediate Command Syntax and Options,
   page 228 for command details.)

   The WSDL files you specify must be concrete WSDLs. This release of the
   AutoMediate Command Line tool does not support abstract WSDLs.

   The AutoMediate Command Line tool executes, creating a SOA project from
   the concrete WSDL.

3. Import your SOA project into TIBCO Business Studio:

   The new project includes the deployment application specification and the application
   archive file that contains the composite application.
Deploying the Generated Deployment Application Archive

After the AutoMediate Command Line creates your SOA project, you can deploy its deployment application archive (.daa file) directly into the runtime environment. See the document *TIBCO ActiveMatrix Administration* for more information.
AutoMediate Command Syntax and Options

Command

Syntax
AutoMediate [-projectName project_name] [-projectDir project_dir] [-serviceHost host_name] [-servicePort port] wsdl location

Description
The AutoMediate command uses concrete WSDL file or directory locations as input and generates a composite application that provides pass-through mediation capabilities for existing web services.

The WSDL URI format must be in EMF URI format.

Options
The following table lists the command-line options.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-projectName</td>
<td>The name of the SOA project:</td>
</tr>
<tr>
<td></td>
<td>• If you specify a name, a single SOA project is created. A single project is created even if you specify multiple WSDL files.</td>
</tr>
<tr>
<td></td>
<td>• If you do not specify a name, all the projects are generated based on the @name attribute of the WSDL definitions element.</td>
</tr>
<tr>
<td></td>
<td>If the @name attribute is not specified in the WSDL, the name of the WSDL file is used.</td>
</tr>
<tr>
<td>-projectDir</td>
<td>The directory where generated projects are to be created.</td>
</tr>
<tr>
<td></td>
<td>If this option is not specified, generated projects are created in the current working directory.</td>
</tr>
<tr>
<td>-serviceHost</td>
<td>The host name for the service endpoint using SOAP over JMS.</td>
</tr>
<tr>
<td></td>
<td>If this option is specified, the AutoMediate Command Line tool overwrites the host name field in the Naming Provider URL field in the generated resource template.</td>
</tr>
<tr>
<td>-servicePort</td>
<td>The port number for the service endpoint using SOAP over JMS.</td>
</tr>
<tr>
<td></td>
<td>If this option is specified, the AutoMediate Command Line tool overwrites the port in the Naming Provider URL in the generated resource template.</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>wsdl location</td>
<td>Specify the location of the source WSDL files:</td>
</tr>
<tr>
<td></td>
<td>• If you use a single WSDL file, specify the name of the source WSDL file or directory location.</td>
</tr>
<tr>
<td></td>
<td>• If you use multiple WSDL files, specify the name of the directory containing the source WSDL files.</td>
</tr>
<tr>
<td></td>
<td>If you do not specify a <code>-projectName</code> when you use multiple WSDL files, the AutoMediate Command Line tool creates one ActiveMatrix SOA project for each concrete WSDL file in the source directory.</td>
</tr>
<tr>
<td>-httpConnector Name</td>
<td>The name of the HTTP connector for services bound using SOAP over HTTP. If this option is not specified, HTTP connector name is the service name plus the port name. It is ignored for web services that are bound using SOAP over JMS.</td>
</tr>
<tr>
<td>-serviceJmsConnectionFactoryJndiName (optional)</td>
<td>The JMS connection factory JNDI name for the resource template that is generated to configure a SOAP over JMS service endpoint. If this field is specified, the AutoMediate Command Line tool will overwrite the connection factory JNDI name in the generated JMS connection factory resource template for the service endpoint.</td>
</tr>
<tr>
<td>-serviceJmsDestinationJndiName (optional)</td>
<td>The JMS destination JNDI name for the resource template that is generated to configure a SOAP over JMS service endpoint. If this field is specified, the AutoMediate Command Line tool will overwrite the destination JNDI name in the generated JMS destination resource template for the service endpoint.</td>
</tr>
<tr>
<td>-refJmsConnectionFactoryJndiName (optional)</td>
<td>The JMS connection factory JNDI name for the resource template that is generated to configure a SOAP over JMS reference endpoint. If this field is specified, the AutoMediate Command Line tool will overwrite the connection factory JNDI name in the generated JMS connection factory resource template for the reference endpoint.</td>
</tr>
<tr>
<td>-refJmsDestinationJndiName (optional)</td>
<td>The JMS destination JNDI name for the resource template that is generated to configure a SOAP over JMS reference endpoint. If this field is specified, the AutoMediate Command Line tool will overwrite the destination JNDI name in the generated JMS destination resource template for the reference endpoint.</td>
</tr>
<tr>
<td>-daaOnly (optional)</td>
<td>Used to generate only the DAA file.</td>
</tr>
</tbody>
</table>
### Example

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>AutoMediate -projectName SOA webservice.wsdl</td>
<td>Automatically mediates one or more web services defined by single WSDL file.</td>
</tr>
<tr>
<td>AutoMediate -projectName SOA webservice.wsdl webservice1.wsdl webservice2.wsdl</td>
<td>Automatically mediates web services.</td>
</tr>
<tr>
<td>AutoMediate c:/tibco/SOAP/wsdls c:/tibco/SOAP/wsdls1 c:/tibco/SOAP/wsdls2</td>
<td></td>
</tr>
<tr>
<td>AutoMediate webservice.wsdl c:/tibco/SOAP/wsdls webservice1.wsdl</td>
<td></td>
</tr>
<tr>
<td>AutoMediate -projectDir C:/tibco/SOA/mediation/workspace webservice.wsdl</td>
<td>Automatically mediates one or more web services and writes generated SOA project to a specific directory.</td>
</tr>
<tr>
<td>AutoMediate -projectDir C:/tibco/SOA/mediation/workspace -serviceHost localhost -servicePort 9897 webservice.wsdl</td>
<td>Automatically mediates one or more web services, generates a SOA project in a specific directory, and updates the host and port for the service.</td>
</tr>
<tr>
<td>AutoMediate -projectDir C:/tibco/SOA/mediation/workspace c:/tibco/wsdls</td>
<td>Automatically mediates web services defined by the WSDL files contained in the specified folder.</td>
</tr>
<tr>
<td>AutoMediate webservice.wsdl -daaOnly -projectDir C:/tibco/SOA/mediation/workspace</td>
<td>Automatically mediates one or more web services and generates the DAA in the specified directory.</td>
</tr>
</tbody>
</table>

### Table 19  AutoMediate Command-Line Exception

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSDLFileNotFoundException</td>
<td>The WSDL location passed to the AutoMediate Command Line tool is invalid.</td>
</tr>
<tr>
<td>NoWSDLServiceDefinedException</td>
<td>The WSDL passed to the AutoMediate Command Line tool does not have any services defined.</td>
</tr>
<tr>
<td></td>
<td>AutoMediate Command Line supports only concrete WSDLs for this add-on pack release.</td>
</tr>
<tr>
<td>Exception</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>NoWSDLServiceBindingException</td>
<td>The port child element of a WSDL service element has a missing binding attribute. Binding must be provided for AutoMediate Command Line to generate a fully functional composite application.</td>
</tr>
<tr>
<td>NoWSDLBindingPortTypeException</td>
<td>A WSDL binding has a missing port type attribute. The port type must be provided for AutoMediate Command Line to generate a fully functional composite application.</td>
</tr>
</tbody>
</table>
AutoMediate ANT Command Syntax and Options

Command

Syntax

amx_eclipse_ant.exe [-DprojectDir=project_dir] [-DprojectName=project_name] [-DserviceHost host_name] [-DservicePort port] [-DwsdlLocation wsdl location] [-DwsdlLocations=wsdl location] [-buildfile build_file]

The AutoMediate ANT command is located in the <TIBCO_HOME>\studio\<version>\eclipse directory

Description

The AutoMediate ANT command generates a composite application that provides pass-through mediation capability for an existing web service or web-services.

The WSDL URI format must be in EMF URI format.

Options

See Table 17, AutoMediate Command Options:, on page 228

Table 20  AutoMediate ANT Command-Line Examples

<table>
<thead>
<tr>
<th>Exception</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>amx_eclipse_ant.exe -DprojectName=SOA webservice.wsdl -buildfile build.xml</td>
<td>Automatically mediates one or more web services defined by single WSDL file.</td>
</tr>
<tr>
<td>amx_eclipse_ant.exe -DprojectName=SOA -DwsdlLocation=webservice.wsdl;webservice1.wsdl;webservice2.wsdl -buildfile build.xml</td>
<td>Automatically mediates web services.</td>
</tr>
<tr>
<td>amx_eclipse_ant.exe -DprojectName=SOA -DwsdlLocation=c:\tibco\SOAP\wsdl1;c:\tibco\SOAP\wsdl2;c:\tibco\SOAP\wsdl3 -buildfile build.xml</td>
<td></td>
</tr>
<tr>
<td>amx_eclipse_ant.exe -DprojectDir=c:/tibco/SOA/mediation/workspace -DwsdlLocation=webservice.wsdl -buildfile build.xml</td>
<td>Automatically mediates one or more web services and writes generated SOA project to a specific directory.</td>
</tr>
<tr>
<td>amx_eclipse_ant.exe -DprojectDir=C:/tibco/SOA/mediation/workspace -serviceHost localhost -servicePort 9897 webservice.wsdl -buildfile build.xml</td>
<td>Automatically mediates one or more web services, generates a SOA project in a specific directory, and updates the host and port for the service.</td>
</tr>
<tr>
<td>Exception</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>amx_eclipse_ant.exe -daaOnly -DprojectDir=c:/tibco/SOA/mediation/workspace -DwsdlLocation=webservice.wsdl -buildfile build.xml</td>
<td>Automatically mediates one or more web services and generates the DAA in the specified directory.</td>
</tr>
</tbody>
</table>

See Table 19, AutoMediate Command-Line Exception, on page 230 for the AutoMediate ANT command-line exceptions.
Appendix B  Developing gXML Applications

The appendix discusses Guideline XML (gXML).

Topics

- Overview on page 236
- Developing gXML Applications on page 237
- gXML Recipes on page 244
Overview

A Generic Java API for XQuery Data Model (XDM) and eXtensible Markup Language (XML) Processing. gXML also provides a cohesive suite of XML processing implementations such as XPath, XSLT, XQuery, Serialization, W3C XML Schema and Validation.

gXML is a way of writing XML code in the Java language. The code that you write to the gXML API can be run against any data model that supports the gXML bridge.

This flexibility offers several benefits:

• Minimizes expensive conversion overhead
• Increases opportunities for performance optimization
• Increases code reuse
• Minimizes risks associated with locking into one Data Model

gXML currently supports Parsing, Serialization, XDM Data Model, XPath 2, XSLT 2 and XQuery, W3C XML Schema and Validation.

• A gXML bridge is provided for org.w3c.dom.Node.
• A gXML bridge for a high performance proprietary implementation is complete but not yet released.
• A gXML bridge for a reference implementation is complete but not yet released. A gXML bridge for AxiOM is in the works.
Developing gXML Applications

This section illustrates one way of using gXML. All gXML processors, including custom processing, run within a GxProcessingContext instance that provides necessary metadata. A GxProcessingContext instance in turn is created through a GxApplication instance.

You must write a class that provides an instance of GxApplication. The best way to do this is to write an abstract class that implements all but the newProcessingContext method of GxApplication. This approach allows you to write your application generically and then inject the choice of parameterization as late as possible for maximum code reuse and flexibility.

This, of course, is not the only way to use gXML. An existing architecture may force the choice of parameterization and create silos of XML processing. The degree of integration in this case may be less than is possible with a homogeneous solution.

Whatever the approach, the best way to use gXML is to write generic, parameterized, and XML processing code whenever possible.

Implementing GxApplication

```java
public abstract class SampleApp<I, U, N extends I, A extends I, S, T, X> extends TestCase implements GxApplication<I, U, N, A, S, T, X>
{
    public Resolver getResolver()
```
```java
{ try {
    return new SampleResolver(new URI("../../plugins/org.gxml.book/resources/foo.xml"));
} catch (final URISyntaxException e) {
    throw new AssertionError(e);
}
}

protected String serialize(final N node, final GxProcessingContext<I, U, N, A, S, T, X> pcx) {
    final GxSerializerFactory<I, U, N, A, S, T, X> sf =
           new SerializerFactory<I, U, N, A, S, T, X>(pcx);

    // Configure for "pretty" printing.
    sf.setIndent(Boolean.TRUE);

    final StringWriter w = new StringWriter();

    final GxSequenceHandler<A, S, T> handler =
           sf.newSerializer(w);

    final GxModel<N, A, S, T> model = pcx.getModel();

    handler.startDocument(null);
    try {
        model.stream(node, true, true, handler);
    } finally {
        handler.endDocument();
    }

    return w.toString();
}

/**
 * Some bridge implementations may use {@link String} directly for symbols. They must make them behave according to
 * symbol semantics (==,toString).
 */
public void assertNodeSymbolSemantics(final N node, final GxProcessingContext<I, U, N, A, S, T, X> pcx) {
    final GxModel<N, A, S, T> model = pcx.getModel();
    final GxNameBridge<S> nameBridge = pcx.getNameBridge();
```
switch (model.getNodeKind(node))
{
    case ELEMENT:
    {
        assertSymbolSemantics(model.getNamespaceURI(node), nameBridge);
        assertSymbolSemantics(model.getLocalName(node), nameBridge);
    }
    case TEXT:
    case DOCUMENT:
    {
    }
    break;
    default:
    {
        throw new AssertionError(model.getNodeKind(node));
    }
}

public void assertSymbolSemantics(final S symbol, final GxNameBridge<S> nameBridge)
{
    PreCondition.assertArgumentNotNull(symbol, "symbol");
    PreCondition.assertArgumentNotNull(nameBridge, "nameBridge");
    assertSame(symbol, nameBridge.symbolize(symbol.toString()));
    assertSame(symbol, nameBridge.symbolize(copy(symbol.toString())));
}

/**
 * Do anything to manufacture a String that is equal, but not identical (the same), as the original.
 * This method has the post-condition that the strings are equal but not the same.
 * @param original
 * The original.
 * @return A copy of the original string.
 */
private String copy(final String original)
{
    final String copy = original.concat("junk").substring(0, original.length());
Implementing GxCatalog

A catalog provides the means to isolate your application from the physical location of file resources. Writing a catalog means implementing the GxCatalog interface so that it maps form the logical locations specified in code or XML resources to the corresponding physical location.

```java
package org.gxml.book.common;

public class SampleCatalog {

}
```

Implementing GxResolver

A resolver takes a base-uri and an href and uses these two values to return a stream.

```java
package org.gxml.book.common;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStream;
import java.net.URI;
import java.net.URISyntaxException;
import java.net.URL;
import org.gxml.xdm.Resolved;
import org.gxml.xdm.Resolver;
import com.tibco.gxml.sa.api.common.util.PreCondition;

public final class SampleResolver implements Resolver {
    final URI baseURI;
```
public SampleResolver(final URI baseURI)
{
    this.baseURI = PreCondition.assertArgumentNotNull(baseURI, "baseURI");
}

/**
 * Convert a URI relative to a base URI into an input source.
 * <p/>
 * This default implementation requires that neither parameter be null, and performs the expected action to retrieve
 * the input source (which may involve network access).
 *<p/>
 * @param baseURI
 * @param location
 * @return a pair of InputStream and resolved URI.
 */
public Resolved<InputStream> resolveInputStream(final URI location) throws IOException
{
    PreCondition.assertArgumentNotNull(location, "uri");
    if (location.isAbsolute())
    {
        return retrieve(location, location);
    }
    else
    {
        PreCondition.assertArgumentNotNull(baseURI, "baseURI");
        final URI base = baseURI.normalize();
        final URI resolved = base.resolve(location);
        return retrieve(location, resolved);
    }
}

private Resolved<InputStream> retrieve(final URI location, final URI uri) throws IOException
{
    PreCondition.assertArgumentNotNull(uri, "uri");
    final URL toRetrieve;
    if (!uri.isAbsolute()) // assume local file
    {
        final File canonFile = new File(uri.toString()).getCanonicalFile();
    }
064         toRetrieve = canonFile.toURI().toURL();
065     } else {
066         toRetrieve = uri.toURL();
067     }
068     if (toRetrieve == null) {
069         throw new FileNotFoundException(uri.toString());
070     }
071     final InputStream stream = toRetrieve.openStream();
072     if (stream == null) {
073         throw new FileNotFoundException(toRetrieve.toString());
074     }
075     try {
076         return new Resolved<InputStream>(location, stream,
077             toRetrieve.toURI());
078     } catch (final URISyntaxException e) {
079         throw new AssertionError(e);
080     }
081 }
082 }
083 catch (final URISyntaxException e) {
084     throw new AssertionError(e);
085 }
086 }
087 }
088 }
089 }
090 }

Injecting DOM

The final task in providing a concrete GxApplication class is to implement the newProcessingContext method on a derived class. You choose the tree, atomic values, metadata and symbols that your application will use. In many cases you can use an off-the-shelf processing context class, but you can also assemble or build your own.

If you are going to use gXML with org.w3c.dom.Node, you have choices for the atomic values that your system will use as well as the metadata implementation. This example uses atomic values that are mostly Java wrapper types and the reference sequence type implementation, SmSequenceType.
import org.gxml.sa.mutable.GxProcessingContextMutable;
import org.gxml.xs.SmMetaBridge;
import org.gxml.xs.SmSequenceType;
import org.w3c.dom.Node;

import com.tibco.gxml.sa.api.common.datatype.StringNameBridge;
import com.tibco.gxml.sa.common.atom.AtomBridge;
import com.tibco.gxml.sa.common.helpers.GxMetaBridgeOnSmMetaBridgeAdapter;
import com.tibco.gxml.sa.common.helpers.SmAtomBridgeOnGxAtomBridgeAdapter;
import com.tibco.gxml.sa.xdm.dom.DomProcessingContext;
import com.tibco.gxml.xs.SmMetaBridgeFactory;

/**
 * Demonstration of constructing a concrete GxApplication(Mutable)
 * implementation using the DOM processing context.
 */
public final class DomValidatingParsingSample extends
    BookValidatingParsingSample<Object, Object, Node, Object, String,
    SmSequenceType<Object, String>, Object> implements
    GxApplicationMutable<Object, Object, Node, Object, String,
    SmSequenceType<Object, String>, Object>
{
    public final GxProcessingContextMutable<Object, Object, Node,
    Object, String, SmSequenceType<Object, String>, Object>
    newProcessingContext()
    {
        // The name bridge is created along with the processing
g        context for maximum concurrency.
        final GxNameBridge<String> nameBridge = new
        StringNameBridge();
        final AtomBridge<String> atomBridge = new
        AtomBridge<String>(nameBridge);
        final SmMetaBridge<Object, String> cache = new
        SmMetaBridgeFactory<Object, String>(new
        SmAtomBridgeOnGxAtomBridgeAdapter<Object,
        String>(atomBridge)).newMetaBridge();
        final GxMetaBridge<Object, String, SmSequenceType<Object,
        String>> metaBridge = new GxMetaBridgeOnSmMetaBridgeAdapter<Object,
        String>(cache, atomBridge);
        final DomProcessingContext<Object, SmSequenceType<Object,
        String>> pcx = new DomProcessingContext<Object, SmSequenceType<Object,
        String>>(this, metaBridge, cache);

        // Set the "owning" processing context on the atom bridge.
        atomBridge.setProcessingContext(pcx);

        // Return the newly constructed processing context.
        return pcx;
    }
}
gXML Recipes

Parsing

Parsing a Character Stream and a Byte Stream

```java
package org.gxml.book.parsing;
import java.io.InputStream;
import java.io.Reader;
import java.io.StringReader;
import java.net.URI;

import org.gxml.book.common.SampleApp;
import org.gxml.sa.GxModel;
import org.gxml.sa.GxNameBridge;
import org.gxml.sa.GxProcessingContext;
import org.gxml.xdm.NodeKind;
import org.gxml.xdm.Resolved;
import org.gxml.xdm.Resolver;

import com.tibco.gxml.sa.common.helpers.GxDocumentBuilder;

{
    public void testCharacterStreamParse() throws Exception
    {
        final GxProcessingContext<I, U, N, A, S, T, X> pcx = newProcessingContext();
        final GxDocumentBuilderFactory<N, S> factory = newDocumentBuilderFactory<>(pcx);
        final GxDocumentBuilderFactory<N, S> factory = newDocumentBuilderFactory<>(pcx);
        final GxDocumentBuilder<N> builder = factory.newDocumentBuilder();
        final String xmlString = "<e>123</e>";
        final URI systemId = new URI("e.xml");
        final Reader characterStream = new StringReader(xmlString);
        final N doc = builder.parse(characterStream, systemId);
        final GxModel<N, A, S, T> model = pcx.getModel();
```
```java
037     assertEquals(NodeKind.DOCUMENT, model.getNodeKind(doc));
038
039     final N e = model.getFirstChildElement(doc);
040     assertEquals(NodeKind.ELEMENT, model.getNodeKind(e));
041     assertEquals("e", model.getLocalNameAsString(e));
042     assertEquals("123", model.getStringValue(e));
043 }
044
045     public void testByteStreamParse() throws Exception
046     {
047         final Resolver resolver = getResolver();
048
049         final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();
050
051         final URI systemId = new URI("email.xml");
052         final Resolved<InputStream> source =
resolver.resolveInputStream(systemId);
053
054         final GxDocumentBuilderFactory<N, S> factory = new
DocumentBuilderFactory<I, U, N, A, S, T, X>(pcx);
055
056         final GxDocumentBuilder<N> builder =
factory.newDocumentBuilder();
057
058         final N document = builder.parse(source.getResource(),
source.getSystemId());
059
060         final GxModel<N, A, S, T> model = pcx.getModel();
061
062     assertEquals(NodeKind.DOCUMENT,
model.getNodeKind(document));
063
064     final N email = model.getFirstChildElement(document);
065     assertEquals(NodeKind.ELEMENT, model.getNodeKind(email));
066     assertEquals("email", model.getLocalNameAsString(email));
067     final GxNameBridge<S> nameBridge = pcx.getNameBridge();
068     final S namespaceURI =
namespaceURI = nameBridge.symbolize("http://www.example.com");
069     final S localName = nameBridge.symbolize("from");
070     final N from = model.getFirstChildElementByName(email,
namespaceURI, localName);
071     assertEquals("Julie", model.getStringValue(from));
072
073     for (final N node : model.getDescendantOrSelfAxis(document))
074     {
075         assertNodeSymbolSemantics(node, pcx);
076     }
077 }
078 }
```
Constructing a Data Model Tree Programmatically

This example demonstrates constructing a tree directly using the fragment builder.

```java
package org.gxml.book.snoopy;

import java.io.IOException;
import java.io.InputStream;
import java.io.StringReader;
import java.io.StringWriter;
import java.net.URI;
import java.net.URISyntaxException;

import javax.xml.namespace.QName;
import javax.xml.parsers.ParserConfigurationException;

import org.gxml.book.common.SampleApp;
import org.gxml.sa.GxException;
import org.gxml.sa.GxFragmentBuilder;
import org.gxml.sa.GxMetaBridge;
import org.gxml.sa.GxModel;
import org.gxml.sa.GxNameBridge;
import org.gxml.sa.GxProcessingContext;
import org.gxml.sa.GxSequenceHandler;
import org.gxml.sa.GxVariantBridge;
import org.gxml.xdm.NodeKind;
import org.gxml.xdm.Resolved;
import org.gxml.xdm.Resolver;
import org.gxml.xs.SmName;

import com.tibco.gxml.sa.api.common.lang.ExprException;
import com.tibco.gxml.sa.api.common.lang.ExprResult;
import com.tibco.gxml.sa.api.common.lang.GxExpr;
import com.tibco.gxml.sa.api.common.lang.GxExprContextDynamicArgs;
import com.tibco.gxml.sa.api.common.lang.GxExprContextStaticArgs;
import com.tibco.gxml.sa.api.common.lang.GxLanguageToolKit;
import com.tibco.gxml.sa.common.helpers.GxDocumentBuilder;
import com.tibco.gxml.sa.processor.serialization.api.GxSerializerFactory;
import com.tibco.gxml.sa.processor.serialization.impl.SerializerFactory;
import com.tibco.gxml.sa.processor.xquery.LanguageToolKit;
import com.tibco.gxml.sa.processor.xslt.GxTransformBuilder;
import com.tibco.gxml.sa.processor.xslt.GxTransformer;
import com.tibco.gxml.sa.processor.xslt.XSLTransformBuilder;
```
043 import com.tibco.gxmlsa.processor.org.exslt.strings.ExsltStringsFunctionGroup;
044
046 {
047     public void testDocumentFromString()
048     {
049         final GxProcessingContext<I, U, N, A, S, T, X> pcx = newProcessingContext();
050         final N document = documentFromString(pcx);
051         final GxSerializerFactory<I, U, N, A, S, T, X> sf = new SerializerFactory<I, U, N, A, S, T, X>(pcx);
052         sf.setIndent(true);
053         final StringWriter sw = new StringWriter();
054         final GxSequenceHandler<A, S, T> serializer = sf.newSerializer(sw);
055         final GxModel<N, A, S, T> model = pcx.getModel();
056         model.stream(document, true, true, serializer);
057         // System.out.println(sw.toString());
058     }
059 }
060
061     public void testFragmentBuilder()
062     {
063         final GxProcessingContext<I, U, N, A, S, T, X> pcx = newProcessingContext();
064         final N document = documentFromEvents(pcx);
065         final GxSerializerFactory<I, U, N, A, S, T, X> sf = new SerializerFactory<I, U, N, A, S, T, X>(pcx);
066         sf.setIndent(true);
067         final StringWriter sw = new StringWriter();
068         final GxSequenceHandler<A, S, T> serializer = sf.newSerializer(sw);
069         final GxModel<N, A, S, T> model = pcx.getModel();
070         model.stream(document, true, true, serializer);
071         // System.out.println(sw.toString());
072     }
073
074 
075 
076 
077 
078 
079 
080 
081 
082 
083 
084 
085
```java
086     // System.out.println(sw.toString());
087 }
088
089     private N documentFromString(final GxProcessingContext<I, U, N, A, S, T, X> pcx)
090     {
091         final String strval = "" + '<?xml version='1.0'
092             encoding='UTF-8'?>" + "<book isbn='0836217462'" + " <title>Being a Dog
093             Is a Full-Time Job</title>" + " <author>Charles M. Schultz</author>" + " <character>
094             <name>Snoopy</name>" + " <since>1950-10-04</since>" + " </character>" + "</book>";
095
096         final GxDocumentBuilderFactory<N, S> factory = new
097             DocumentBuilderFactory<I, U, N, A, S, T, X>(pcx);
098         final GxDocumentBuilder<N> builder =
099             factory.newDocumentBuilder();
100     }
101         try
102         {
103             return builder.parse(new StringReader(strval), null);
104         }
105         catch (final IOException e)
106         {
107             throw new AssertionError();
108         }
109     }
110
111     private N documentFromEvents(final GxProcessingContext<I, U, N, A, S, T, X> pcx)
112     {
113         final GxNameBridge<S> nameBridge = pcx.getNameBridge();
114         final S NULL_NS_URI = nameBridge.empty();
115         final S BOOK = nameBridge.symbolize("book");
116         final S ISBN = nameBridge.symbolize("isbn");
117         final S TITLE = nameBridge.symbolize("title");
118         final S AUTHOR = nameBridge.symbolize("author");
119         final S CHARACTER = nameBridge.symbolize("character");
120         final S NAME = nameBridge.symbolize("name");
121         final S SINCE = nameBridge.symbolize("since");
122         final GxFragmentBuilder<N, A, S, T> builder =
123             pcx.newFragmentBuilder();
124         try
125         {
126             // Note: Using try...finally not only ensures that elements
127             // get closed when errors
128             // occur, it also helps to remind you to end elements and
129             // makes the levels in
130             // the XML more obvious.
131             builder.startDocument(null);
132             try
133             {
134                 // System.out.println(sw.toString());
135             }
136         }
137         finally
138         {
139             builder.endDocument();
140         }
141     }
```
builder.startElement(NULL_NS_URI, BOOK, "", null);
try {
    builder.attribute(NULL_NS_URI, ISBN, "", "0836217462");
    builder.startElement(NULL_NS_URI, TITLE, "", null);
    try {
        builder.text("Being a Dog Is a Full-Time Job");
    } finally {
        builder.endElement();
    }
    builder.startElement(NULL_NS_URI, AUTHOR, "", null);
    try {
        builder.text("Charles M. Schultz");
    } finally {
        builder.endElement();
    }
    builder.startElement(NULL_NS_URI, CHARACTER, "", null);
    try {
        builder.startElement(NULL_NS_URI, NAME, "", null);
        try {
            builder.text("Snoopy");
        } finally {
            builder.endElement();
        }
        builder.startElement(NULL_NS_URI, SINCE, "", null);
        try {
            builder.text("1950-10-04");
        } finally {
            builder.endElement();
        }
    } finally {
        builder.endElement();
    }
}
173     {
174         builder.endElement();
175     }
176     }
177     finally
178     {
179         builder.endElement();
180     }
181     }
182     finally
183     {
184         builder.endDocument();
185     }
186     return builder.getNodes().get(0);
187     }
188     }
189     public void testExample() throws ParserConfigurationException,
190         IOException, GxException, ExprException, URISyntaxException
191     {
192         final GxProcessingContext<I, U, N, A, S, T, X> pcx =
193             newProcessingContext();
194             final Resolver resolver = getResolver();
195             final URI xmlSystemId = new URI("hotel.xml");
196             final Resolved<InputStream> xmlInput =
197                 resolver.resolveInputStream(xmlSystemId);
198             final GxDocumentBuilderFactory<N, S> f = new
199                 DocumentBuilderFactory<I, U, N, A, S, T, X>(pcx);
200                 final GxDocumentBuilder<N> builder = f.newDocumentBuilder();
201                 final N document = builder.parse(xmlInput.getResource(),
202                     xmlInput.getSystemId());
203                 final URI xslSystemId = new URI("hotel.xsl");
204                 final Resolved<InputStream> xslInput =
205                     resolver.resolveInputStream(xslSystemId);
206                 final GxTransformBuilder<I, U, N, A, S, T, X> compiler = new
207                     XSLTransformBuilder<I, U, N, A, S, T, X>(pcx);
208                     // poem.xsl uses version="2.0", but we want to use XPath 1.0
209                     // compatibility mode
210                     // so that arguments to functions are converted etc.
211                     compiler.setCompatibleMode(true);
212                     final GxTransform<I, U, N, A, S, T, X> compiled =
213                         compiler.prepareTransform(xslInput.getResource(),
214                             xslInput.getSystemId());
final GxSerializerFactory<I, U, N, A, S, T, X> sf = new SerializerFactory<I, U, N, A, S, T, X>(pcx);

// TODO: Extract the configuration?
// compiled.configure(sf);

sf.setIndent(true);

final StringWriter w = new StringWriter();

final GxSequenceHandler<A, S, T> handler = sf.newSerializer(w);

final GxTransformer<I, U, N, A, S, T, X> transformer = compiled.newTransformer();

transformer.transform(document, pcx, handler);
}

public void testVariableBinding() throws ParserConfigurationException, IOException, GxException, ExprException, URISyntaxException
{
    final GxProcessingContext<I, U, N, A, S, T, X> pcx = newProcessingContext();

    final Resolver resolver = getResolver();

    final URI xslSystemId = new URI("email.xsl");
    final Resolved<InputStream> xslInput = resolver.resolveInputStream(xslSystemId);

    final GxTransformBuilder<I, U, N, A, S, T, X> compiler = new XSLTransformBuilder<I, U, N, A, S, T, X>(pcx);

    final GxTransform<I, U, N, A, S, T, X> compiled = compiler.prepareTransform(xslInput.getResource(), xslInput.getSystemId());

    final GxTransformer<I, U, N, A, S, T, X> transformer = compiled.newTransformer();

    final GxNameBridge<S> nameBridge = pcx.getNameBridge();
    final SmName<S> varName = nameBridge.name(new QName("to"));

    final GxVariantBridge<I, N, A, X> valueBridge = pcx.getVariantBridge();
    final X value = valueBridge.stringValue("David");

    transformer.bindVariableValue(varName, value);
    transformer.bindVariableValue(nameBridge.name(new QName("http://www.example.com", "from")), valueBridge.stringValue("Julie"));
final N documentNode = transformer.transform(null, pcx);
final GxModel<N, A, S, T> model = pcx.getModel();
assertEquals(NodeKind.DOCUMENT, model.getNodeKind(documentNode));
final N email = model.getFirstChildElement(documentNode);
final N to = model.getFirstChildElementByName(email, nameBridge.symbolize("http://www.example.com"), nameBridge.symbolize("to"));
assertEquals("David", model.getStringValue(to));
final N from = model.getFirstChildElementByName(email, null, nameBridge.symbolize("from"));
assertEquals("Julie", model.getStringValue(from));
final N again = model.getFirstChildElementByName(email, nameBridge.symbolize("http://www.example.com"), null);
assertEquals("David", model.getStringValue(again));

public void testExternalFunctions() throws 
ParserConfigurationException, IOException, GxException, ExprException, URISyntaxException
{
final GxProcessingContext<I, U, N, A, S, T, X> pcx = newProcessingContext();
final Resolver resolver = getResolver();
final URI xmlSystemId = new URI("exslt.xml");
final Resolved<InputStream> xmlInput = resolver.resolveInputStream(xmlSystemId);
final GxDocumentBuilderFactory<N, S> f = new DocumentBuilderFactory<I, U, N, A, S, T, X>(pcx);
final GxDocumentBuilder<N> builder = f.newDocumentBuilder();
final N document = builder.parse(xmlInput.getResource(), xmlInput.getSystemId());
final URI xslSystemId = new URI("exslt.xsl");
final Resolved<InputStream> xslInput = resolver.resolveInputStream(xslSystemId);
final GxTransformBuilder<I, U, N, A, S, T, X> compiler = new XSLTransformBuilder<I, U, N, A, S, T, X>(pcx);
final String namespaceURI = "http://exslt.org/strings";
final ExsltStringsFunctionGroup<I, U, N, A, S, T, X> functions = new ExsltStringsFunctionGroup<I, U, N, A, S, T, X>(namespaceURI, pcx);
compiler.setFunctionSigns(namespaceURI, functions);
compiler.setFunctionImpls(namespaceURI, functions);

final GxTransform<I, U, N, A, S, T, X> compiled =
compiler.prepareTransform(xslInput.getResource(),
xslInput.getSystemId());

final GxSerializerFactory<I, U, N, A, S, T, X> sf = new
SerializerFactory<I, U, N, A, S, T, X>(pcx);

// TODO: Extract the configuration.
// compiled.configure(sf);

sf.setIndent(true);

final StringWriter w = new StringWriter();

final GxSequenceHandler<A, S, T> handler =
sf.newSerializer(w);

final GxTransformer<I, U, N, A, S, T, X> transformer =
compiled.newTransformer();

transformer.transform(document, pcx, handler);

// System.out.println(w.toString());
}

public void testHotel() throws ParserConfigurationException,
IOException, GxException, ExprException, URISyntaxException
{ final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();

final Resolver resolver = getResolver();

final URI xmlSystemId = new URI("hotel.xml");
final Resolved<InputStream> xmlInput =
resolver.resolveInputStream(xmlSystemId);

final GxDocumentBuilderFactory<N, S> f = new
DocumentBuilderFactory<I, U, N, A, S, T, X>(pcx);

final GxDocumentBuilder<N> builder = f.newDocumentBuilder();

final N document = builder.parse(xmlInput.getResource(),
xmlInput.getSystemId());

final URI xslSystemId = new URI("hotel.xsl");
final Resolved<InputStream> xslInput =
resolver.resolveInputStream(xslInput);
final GxTransformBuilder<I, U, N, A, S, T, X> compiler = new XSLTransformBuilder<I, U, N, A, S, T, X>(pcx);

final GxTransform<I, U, N, A, S, T, X> compiled = compiler.prepareTransform(xslInput.getResource(), xslInput.getSystemId());

final GxSerializerFactory<I, U, N, A, S, T, X> sf = new SerializerFactory<I, U, N, A, S, T, X>(pcx);

// TODO: Extract the configuration.
// compiled.configure(sf);

sf.setIndent(true);

final StringWriter w = new StringWriter();

final GxSequenceHandler<A, S, T> handler = sf.newSerializer(w);

final GxTransformer<I, U, N, A, S, T, X> transformer = compiled.newTransformer();

final GxNameBridge<S> nameBridge = pcx.getNameBridge();

final SmName<S> varName = nameBridge.name(new QName("MessageData"));

final GxVariantBridge<I, N, A, X> valueBridge = pcx.getVariantBridge();

final X value = valueBridge.node(document);

transformer.bindVariableValue(varName, value);

transformer.transform(null, pcx, handler);

// System.out.println(w.toString());

public void testHelloWorld() throws Exception {
    final GxProcessingContext<I, U, N, A, S, T, X> pcx = new ProcessingContext();
    final GxNameBridge<S> nameBridge = pcx.getNameBridge();

    final GxLanguageToolKit<I, U, N, A, S, T, X> xtk = new LanguageToolKit<I, U, N, A, S, T, X>(pcx);

    final GxExprContextStaticArgs<I, U, N, A, S, T, X> senv = xtk.newStaticContextArgs();

    final String NAMESPACE = "http://www.peanuts.com";

    senv.getInScopeNamespaces().declarePrefix("nuts", nameBridge.symbolize(NAMESPACE));
372         final SnoopyFunctionGroup<I, U, N, A, S, T, X>
373             peanutsFunctionGroup = new SnoopyFunctionGroup<I, U, N, A, S, T, X>(NAMESPACE, pcx);
374         senv.setFunctionSigns(NAMESPACE, peanutsFunctionGroup);
375         senv.setFunctionImpls(NAMESPACE, peanutsFunctionGroup);
376         final GxMetaBridge<A, S, T> metaBridge =
377             pcx.getMetaBridge();
378         final ExprResult<I, U, N, A, S, T, X> prepared =
379             xtk.prepare("nuts:GetVariableProperty('foo','bar')", metaBridge.emptyType(), senv);
380         final GxExpr<I, U, N, A, S, T, X> expr = prepared.getExpr();
381         final GxExprContextDynamicArgs<I, U, N, A, S, T, X> darg =
382             xtk.newDynamicContextArgs();
383         final String strval = expr.stringFunction(xtk.emptyFocus(),
384             darg, pcx);
385         assertEquals("Bingo!", strval);
386     }
387 }

Validating

001 package org.gxml.book.parsing;
002
003 import java.io.InputStream;
004 import java.net.URI;
005
006 import javax.xml.namespace.QName;
007
008 import org.gxml.book.common.SampleApp;
009 import org.gxml.sa.GxApplication;
010 import org.gxml.sa.GxAtomBridge;
011 import org.gxml.sa.GxModel;
012 import org.gxml.sa.GxNameBridge;
013 import org.gxml.sa.GxProcessingContext;
014 import org.gxml.xdm.Resolved;
015 import org.gxml.xdm.Resolver;
016 import org.gxml.xs.SmComponentBag;
017 import org.gxml.xs.SmExceptionCatcher;
018 import org.gxml.xs.SmMetaLoadArgs;
019 import org.gxml.xs.SmName;
020
import com.tibco.gxml.sa.common.helpers.GxDocumentBuilder;
import com.tibco.gxml.sa.common.helpers.SmAtomBridgeOnGxAtomBridgeAdapter;
import com.tibco.gxml.xs.W3cXmlSchemaParser;

{
    public void testValidatingParse() throws Exception
    {
        final GxApplication<I, U, N, A, S, T, X> app = this;

        final Resolver resolver = app.getResolver();

        final SmMetaLoadArgs args = new SmMetaLoadArgs();

        final SmExceptionCatcher errors = new SmExceptionCatcher();

        final GxProcessingContext<I, U, N, A, S, T, X> pcx =
            app.newProcessingContext();

        final Resolved<InputStream> resource =
            getResolver().resolveInputStream(new URI("email.xsd"));

        final W3cXmlSchemaParser<A, S> parser = new
            W3cXmlSchemaParser<A, S>(new SmAtomBridgeOnGxAtomBridgeAdapter<A, S>(pcx.getAtomBridge()));

        final SmComponentBag<A, S> components =
            parser.parse(resource.getLocation(), resource.getResource(),
                resource.getSystemId(), errors, args, pcx);

        pcx.register(components);

        pcx.lock();

        final GxNameBridge<S> nameBridge = pcx.getNameBridge();

        assertEquals(0, errors.size());

        final URI xmlURI = new URI("email.xml");

        final Resolved<InputStream> xmlInput =
            resolver.resolveInputStream(xmlURI);

        final GxDocumentBuilderFactory<N, S> factory = new
            DocumentBuilderFactory<I, U, N, A, S, T, X>(pcx);

        // Enable validation of the XML input.
        factory.setValidating(true, nameBridge.name(new
            QName("http://www.example.com", "email"));

        }
final GxDocumentBuilder<N> builder = factory.newDocumentBuilder();

// TODO: Need to catch errors...
// builder.setExceptionHandler(errors);

final N doc = builder.parse(xmlInput.getResource(), xmlInput.getSystemId());

assertEquals(0, errors.size());

// System.out.println(serialize(doc, pcx));

final GxModel<N, A, S, T> model = pcx.getModel();
final GxAtomBridge<A, S> atomBridge = pcx.getAtomBridge();

final N email = model.getFirstChildElement(doc);
final S namespaceURI = nameBridge.symbolize("http://www.example.com");
final N sent = model.getFirstChildElementByName(email, namespaceURI, nameBridge.symbolize("sent"));

assertNotNull("model.getFirstChildElementByName", sent);
final SmName<S> typeName = model.getTypeName(sent);
assertNotNull("model.getTypeName", typeName);
assertEquals("dateTime", typeName.toQName().getLocalPart());
final A dateTime = model.getTypedValue(sent).get(0);

assertTrue(metaBridge.sameAs(metaBridge.handle(pcx.getTypeDefinition(type)),
// metaBridge.getType(SmNativeType.DATETIME)));

assertEquals("2008-03-23T14:49:30-05:00", atomBridge.getC14NForm(dateTime));

} }

    public void testBooksByNealStephenson() throws Exception {
        final Resolver resolver = getResolver();

        final GxProcessingContext<I, U, N, A, S, T, X> pcx = newProcessingContext();

        final URI systemId = new URI("books.xml");
        final Resolved<InputStream> source = resolver.resolveInputStream(systemId);

        final GxDocumentBuilderFactory<N, S> factory = new DocumentBuilderFactory<>(pcx);

        final GxDocumentBuilder<N> builder = factory.newDocumentBuilder();

        final N doc = builder.parse(source.getResource(), source.getSystemId());

        final GxModel<N, A, S, T> model = pcx.getModel();

        final GxNameBridge<S> nameBridge = pcx.getNameBridge();

        final S namespaceURI = nameBridge.symbolize("http://www.example.com/books");

        final N inventory = model.getFirstChildElementByName(doc, namespaceURI, nameBridge.symbolize("inventory"));

        for (final N book : model.getChildElementsByName(inventory, namespaceURI, nameBridge.symbolize("book"))) {
            boolean found = false;

            for (final N author : model.getChildElementsByName(book, namespaceURI, nameBridge.symbolize("author"))) {
                if (model.getStringValue(author).equals("Neal Stephenson")) {
                    found = true;
                    break;
                }
            }
        }
    }
}
053         } 
054     
055     if (found) 
056     { 
057         final N title = 
058             model.getFirstChildElementByName(book, namespaceURI, 
059                 nameBridge.symbolize("title")); 
060             System.out.println(model.getStringValue(title)); 
061     } 
062 } 
063 
064     public void testPurchaseOrder() throws Exception 
065     { 
066         final Resolver resolver = getResolver(); 
067         final GxProcessingContext<I, U, N, A, S, T, X> pcx = 
068             newProcessingContext(); 
069         final GxModel<N, A, S, T> model = pcx.getModel(); 
070         final GxNameBridge<S> nameBridge = pcx.getNameBridge(); 
071         final URI systemId = new URI("PurchaseOrder.xml"); 
072         final Resolved<InputStream> source = 
073             resolver.resolveInputStream(systemId); 
074         final GxDocumentBuilderFactory<N, S> factory = new 
075             DocumentBuilderFactory<I, U, N, A, S, T, X>(pcx); 
076         final GxDocumentBuilder<N> builder = 
077             factory.newDocumentBuilder(); 
078         final N po = builder.parse(source.getResource(), 
079             source(SystemId())); 
080         final N root = model.getFirstChildElement(po); 
081         final N items = model.getFirstChildElementByName(root, null, 
082             nameBridge.symbolize("items")); 
083         double total = 0; 
084         for (final N item : model.getChildElementsByName(items, 
085             nameBridge.symbolize("item")) 
086             { 
087                 System.out.println("partNum:" + 
088                     model.getAttributeStringValue(item, nameBridge.empty(), 
089                         nameBridge.symbolize("partNum"))); 
090                 final N price = model.getFirstChildElementByName(item, 
091                     null, nameBridge.symbolize("USPrice")); 
092                 total += 
093                     Double.valueOf(model.getStringValue(price)).doubleValue(); 
094             }
System.out.println("Grand total = " + total);
}
}

Mutation

package org.gxml.book.mutable;
import java.math.BigDecimal;
import javax.xml.XMLConstants;
import org.gxml.sa.GxAtomBridge;
import org.gxml.sa.GxNameBridge;
import org.gxml.sa.mutable.GxModelMutable;
import org.gxml.sa.mutable.GxProcessingContextMutable;
import org.gxml.xdm.NodeKind;
/**
 * This sample illustrates the use of the optional mutability API.
 *
 * @author dholmes
 *
 * @param <I>
 * @param <U>
 * @param <N>
 * @param <A>
 * @param <S>
 * @param <T>
 * @param <X>
 */
{
    /**
     * This is a test of basic mutability through the optional mutability API.
     *
     * @Line 2
     * @Line 3
     * @Line 4 // OK
     *
     */
    public void testIntroduction() throws Exception
    {
        final GxProcessingContextMutable<I, U, N, A, S, T, X> pcx = newProcessingContext();
        final GxAtomBridge<A, S> atomBridge = pcx.getAtomBridge();
        final GxNameBridge<S> nameBridge = pcx.getNameBridge();
final N documentNode = pcx.newDocument();

final GxModelMutable<N, A, S, T> model = pcx.getModel();

assertEquals(NodeKind.DOCUMENT, model.getNodeKind(documentNode));

final N owner = model.getOwner(documentNode);

assertTrue(model.isSameNode(documentNode, owner));

final S namespaceURI = nameBridge.symbolize("http://www.example.com");
final S localName = nameBridge.symbolize("foo");
final String prefix = "x";
final N documentElement = model.createElement(owner, namespaceURI, localName, prefix);

model.appendChild(documentNode, documentElement);
model.setNamespace(documentElement, prefix, namespaceURI);
model.setAttribute(documentElement, nameBridge.empty(), nameBridge.symbolize("version"), XMLConstants.DEFAULT_NS_PREFIX, atomBridge.wrapAtom(atomBridge.createDecimal(BigDecimal.valueOf(2.7))));

model.appendChild(documentElement, model.createText(owner, "Hello"));
model.appendChild(documentElement, model.createText(owner, " "));
model.appendChild(documentElement, model.createText(owner, "World"));
model.appendChild(documentElement, model.createText(owner, "!"));

model.normalize(documentNode);

@SuppressWarnings("unused")
final String strval = serialize(documentNode, pcx);
//System.out.println(strval);
Serialization

```java
001 package org.gxml.book.serialization;
002
003 import java.io.StringWriter;
004
005 import javax.xml.namespace.QName;
006
007 import org.gxml.book.common.SampleApp;
008 import org.gxml.sa.GxModel;
009 import org.gxml.sa.GxProcessingContext;
010 import org.gxml.sa.GxSequenceHandler;
011 import org.gxml.xdm.Emulation;
012
013 import com.tibco.gxml.sa.processor.serialization.api.GxDocumentSerializer;
014 import com.tibco.gxml.sa.processor.serialization.api.GxDocumentSerializerFactory;
015 import com.tibco.gxml.sa.processor.serialization.api.GxSerializerFactory;
017 import com.tibco.gxml.sa.processor.serialization.impl.SerializerFactory;
018
020 {
021     public void exampleUsingDocumentSerializer(final N node, final GxProcessingContext<I, U, N, A, S, T, X> pcx)
022     {
023         final GxDocumentSerializerFactory<N, S> sf = new DocumentSerializerFactory<I, U, N, A, S, T, X>(pcx);
024
025             // Configure for "pretty" printing.
026             sf.setIndent(Boolean.TRUE);
027             sf.setMethod(new QName("xml"));
028             sf.setOmitXmlDeclaration(false);
029
030             final StringWriter sw = new StringWriter();
031             final GxDocumentSerializer<N> serializer = sf.newDocumentSerializer(sw);
032             serializer.serialize(node);
033             System.out.print(sw.toString());
034         }
035     }
```
public void exampleUsingSequenceHandler(final N node, final 
GxProcessingContext<I, U, N, A, S, T, X> pcx) 
{ 
    final GxSerializerFactory<I,U,N,A, S,T,X> sf = new 
SerializerFactory<I, U, N, A, S, T, X>(pcx); 
    // Configure for "pretty" printing. 
    sf.setIndent(Boolean.TRUE); 
    sf.setMethod(new QName("xml")); 
    sf.setOmitXmlDeclaration(false); 
    sf.setEmulation(Emulation.C14N); 
    final StringWriter sw = new StringWriter(); 
    final GxSequenceHandler<A, S, T> serializer = 
sf.newSerializer(sw); 
    final GxModel<N, A, S, T> model = pcx.getModel(); 
    model.stream(node, true, true, serializer); 
    System.out.print(sw.toString()); 
}

**XPath**

package org.gxml.book.xpath;

import org.gxml.book.common.SampleApp;
import org.gxml.sa.GxMetaBridge;
import org.gxml.sa.GxNameBridge;
import org.gxml.sa.GxProcessingContext;
import org.gxml.sa.GxVariantBridge;
import org.gxml.xdm.Emulation;
import org.gxml.xs.SmName;
import org.gxml.xs.SmNativeType;
import com.tibco.gxml.sa.api.common.lang.ExprResult;
import com.tibco.gxml.sa.api.common.lang.GxExpr;
import com.tibco.gxml.sa.api.common.lang.GxExprContextDynamicArgs;
import com.tibco.gxml.sa.api.common.lang.GxExprContextStaticArgs;
import com.tibco.gxml.sa.api.common.lang.GxFocus;
import com.tibco.gxml.sa.api.common.lang.GxLanguageToolKit;
021 public abstract class XPathSample<I, U, N extends I, A extends I, S, T, X> extends SampleApp<I, U, N, A, S, T, X> {
022
    public void testGettingStarted() throws Exception {
023        final GxProcessingContext<I, U, N, A, S, T, X> pcx = newProcessingContext();
024
025        // For demonstration purposes, register the language toolkit with the processing context.
026        pcx.register("xyz", new LanguageToolKit<I, U, N, A, S, T, X>(pcx));
027
028        @SuppressWarnings("unchecked")
029        // Immediately get back the registered processor.
030        GxLanguageToolKit<I, U, N, A, S, T, X> xtk = pcx.getProcessor("xyz", GxLanguageToolKit.class);
031
032        final GxExprContextStaticArgs<I, U, N, A, S, T, X> sarg = xtk.newStaticContextArgs();
033
034        final ExprResult<I, U, N, A, S, T, X> prepared =
035            xtk.prepare("concat('Hello', ', ', 'World', '!!')", metaBridge.emptyType(), sarg);
036
037        final GxExprContextDynamicArgs<I, U, N, A, S, T, X> darg =
038            xtk.newDynamicContextArgs();
039
040        final GxExpr<I, U, N, A, S, T, X> expr = prepared.getExpr();
041
042        final String strval = expr.stringFunction(xtk.emptyFocus(), darg, pcx);
043
044        assertEquals("Hello, World!", strval);
045    }
046
047    public void testBindingVariables() throws Exception {
048        final GxProcessingContext<I, U, N, A, S, T, X> pcx = newProcessingContext();
049
050        final GxLanguageToolKit<I, U, N, A, S, T, X> xtk = new LanguageToolKit<I, U, N, A, S, T, X>(pcx);
051
052        final GxExprContextStaticArgs<I, U, N, A, S, T, X> statArgs =
053            xtk.newStaticContextArgs();
054
055        statArgs.setEmulation(Emulation.MODERN);
056
057        final GxNameBridge<S> nameBridge = pcx.getNameBridge();
058        final SmName<S> varName = new SmName<S>(nameBridge.symbolize("x"), nameBridge);
final GxMetaBridge<A, S, T> metaBridge =
    pcx.getMetaBridge();
statArgs.bindVariableType(varName,
    metaBridge.getType(SmNativeType.STRING));

final String es = "concat('Hello', ', ', $x, '!')";
final T sfocus = metaBridge.emptyType();

final ExprResult<I, U, A, S, T, X> prepared =
    xtk.prepare(es, sfocus, statArgs);

final GxExprContextDynamicArgs<I, U, A, S, T, X> dynArgs =
    xtk.newDynamicContextArgs();
dynArgs.setEmulation(Emulation.MODERN);

final GxVariantBridge<I, A, X> valueBridge =
    pcx.getVariantBridge();
final X value = valueBridge.stringValue("World");
dynArgs.bindVariableValue(varName, value);

final GxExpr<I, U, A, S, T, X> expr = prepared.getExpr();
final GxFocus<I> dfocus = xtk.emptyFocus();
final String strval = expr.stringFunction(dfocus, dynArgs,
    pcx);
assertEquals("Hello, World!", strval);
}

public void testEXSLT() throws Exception
{
    final GxProcessingContext<I, U, A, S, T, X> pcx =
        newProcessingContext();
    final GxNameBridge<S> nameBridge = pcx.getNameBridge();

    final GxLanguageToolKit<I, U, A, S, T, X> xtk = new
        LanguageToolKit<I, U, A, S, T, X>(pcx);

    final GxExprContextStaticArgs<I, U, A, S, T, X> sarg =
        xtk.newStaticContextArgs();
    sarg.getInScopeNamespaces().declarePrefix("math",
        nameBridge.symbolize("http://exslt.org/math"));
    final ExsltMathFunctionGroup<I, U, A, S, T, X>
        ("http://exslt.org/math", pcx);
    sarg.setFunctionSigns("http://exslt.org/math",
        exsltMathFunctionGroup);
    // The function implementations can be provided now or just
    prior to execution.
    sarg.setFunctionImps("http://exslt.org/math",
        exsltMathFunctionGroup);
final GxMetaBridge<A, S, T> metaBridge = pcx.getMetaBridge();

final ExprResult<I, U, N, A, S, T, X> prepared = xtk.prepare("math:exp(1)", metaBridge.emptyType(), sarg);

final GxExpr<I, U, N, A, S, T, X> expr = prepared.getExpr();

final GxExprContextDynamicArgs<I, U, N, A, S, T, X> darg = xtk.newDynamicContextArgs();
// Here we also (redundantly) provide the function implementations just prior to execution.
darg.setFunctionImpls("http://exslt.org/math", exsltMathFunctionGroup);

final String strval = expr.stringFunction(xtk.emptyFocus(), darg, pcx);
assertEquals("2.7182818284590455", strval);
}

public void testExpressionType() throws Exception
{
final GxProcessingContext<I, U, N, A, S, T, X> pcx = newProcessingContext();
final GxLanguageToolKit<I, U, N, A, S, T, X> xtk = new LanguageToolKit<I, U, N, A, S, T, X>(pcx);
final GxExprContextStaticArgs<I, U, N, A, S, T, X> sarg = xtk.newStaticContextArgs();
final GxMetaBridge<A, S, T> metaBridge = pcx.getMetaBridge();

final ExprResult<I, U, N, A, S, T, X> prepared = xtk.prepare(""Hello"", metaBridge.emptyType(), sarg);
/* final GxExpr<I, U, N, A, S, T, X> expr = */prepared.getExpr();
/* final GxExprInfo<T> info = */prepared.getInfo();
}

XSLT

package org.gxml.book.xslt;

import java.io.IOException;
import java.io.InputStream;
import java.io.StringReader;
import java.io.StringWriter;
import java.net.URI;
import java.net.URISyntaxException;

import javax.xml.namespace.QName;
import javax.xml.parsers.ParserConfigurationException;

import org.gxml.book.common.SampleApp;
import org.gxml.sa.GxException;
import org.gxml.sa.GxMetaBridge;
import org.gxml.sa.GxModel;
import org.gxml.sa.GxNameBridge;
import org.gxml.sa.GxProcessingContext;
import org.gxml.sa.GxSequenceHandler;
import org.gxml.sa.GxVariantBridge;
import org.gxml.xdm.NodeKind;
import org.gxml.xdm.Resolved;
import org.gxml.xdm.Resolver;
import org.gxml.xs.SmName;
import org.gxml.xs.SmNativeType;

import com.tibco.gxml.sa.api.common.lang.ExprException;
import com.tibco.gxml.sa.common.helpers.GxDocumentBuilder;
import com.tibco.gxml.sa.processor.serialization.api.GxSerializerFactory;
import com.tibco.gxml.sa.processor.serialization.impl.SerializerFactory;
import com.tibco.gxml.sa.processor.xslt.GxTransform;
import com.tibco.gxml.sa.processor.xslt.GxTransformBuilder;
import com.tibco.gxml.sa.processor.xslt.GxTransformer;
import com.tibco.gxmlsa.processor.org.exslt.strings.ExsltStringsFunctionGroup;

{
    public void testExample() throws ParserConfigurationException, IOException, GxException, ExprException, URISyntaxException
    {
        final GxProcessingContext<I, U, N, A, S, T, X> pcx = new ProcessingContext();
        final GxMetaBridge<A, S, T> metaBridge = pcx.getMetaBridge();
        final GxNameBridge<S> nameBridge = pcx.getNameBridge();
        final Resolver resolver = getResolver();
        final URI xmlSystemId = new URI("hotel.xml");
final Resolved<InputStream> xmlInput = resolver.resolveInputStream(xmlSystemId);

final GxDocumentBuilderFactory<N, S> f = new DocumentBuilderFactory<I, U, N, A, S, T, X>(pcx);
f.setIgnoreComments(false);

final GxDocumentBuilder<N> builder = f.newDocumentBuilder();

final N document = builder.parse(xmlInput.getResource(), xmlInput.getSystemId());

final URI xslSystemId = new URI("hotel.xsl");
final Resolved<InputStream> xslInput = resolver.resolveInputStream(xslSystemId);

final GxTransformBuilder<I, U, N, A, S, T, X> compiler = new XSLTransformBuilder<I, U, N, A, S, T, X>(pcx);

compiler.setCompatibleMode(true);
// compiler.setRestrictedMode(true); // XSLT 2.0 subset for mapper.

// Specify the static type for the context item:
// document-node(element(*,xs:untyped))
final T documentType = metaBridge.documentType(metaBridge.elementType(new SmName<S>(null, null, nameBridge), metaBridge.getType(SmNativeType.UNTYPED), false));

compiler.setFocus(documentType);

final GxTransform<I, U, N, A, S, T, X> compiled = compiler.prepareTransform(xslInput.getResource(), xslInput.getSystemId());

final GxSerializerFactory<I, U, N, A, S, T, X> sf = new SerializerFactory<I, U, N, A, S, T, X>(pcx);

// TODO: Extract output configuration.
// compiled.configure(sf);

sf.setIndent(true);

final StringWriter w = new StringWriter();
final GxSequenceHandler<A, S, T> handler = sf.newSerializer(w);
final GxTransformer<I, U, N, A, S, T, X> transformer = compiled.newTransformer();

transformer.transform(document, pcx, handler);

@SuppressWarnings("unused")
final String s = w.toString();

// System.out.println(s);
}

@SuppressWarnings("unused")
private void bar(final GxProcessingContext<I, U, N, A, S, T, X> pcx)
{
    try
    {
        final GxTransformBuilder<I, U, N, A, S, T, X> builder =
         new XSLTransformBuilder<I, U, N, A, S, T, X>(pcx);

        final GxTransform<I, U, N, A, S, T, X> transform =
         builder.prepareTransform(new StringReader("<x xsl:version='1.0' xmlns:xsl='http://www.w3.org/1999/XSL/Transform'></x>"), new URI(""));

        final GxTransformer<I, U, N, A, S, T, X> transformer =
         transform.newTransformer();

        final N document = transformer.transform(null, pcx);

        final GxModel<N, A, S, T> model = pcx.getModel();

        final N element = model.getFirstChild(document);

        final String name = model.getLocalNameAsString(element);

        // System.out.println("XSLT: " + name);
    }
    catch (final Throwable e)
    {
        e.printStackTrace();
    }
}

public void skipVariableBinding() throws
ParserConfigurationException, IOException, GxException, ExprException, URISyntaxException
{
    final GxProcessingContext<I, U, N, A, S, T, X> pcx =
     newProcessingContext();

    final Resolver resolver = getResolver();

    final URI xslSystemId = new URI("email.xsl");

    final Resolved<InputStream> xslInput =
     resolver.resolveInputStream(xslSystemId);

    final GxTransformBuilder<I, U, N, A, S, T, X> compiler =
     new XSLTransformBuilder<I, U, N, A, S, T, X>(pcx);
132     final GxTransform<I, U, N, A, S, T, X> compiled = compiler.prepareTransform(xslInput.getResource(), xslInput.getSystemId());
133
134     final GxTransformer<I, U, N, A, S, T, X> transformer = compiled.newTransformer();
135
136     final GxNameBridge<S> nameBridge = pcx.getNameBridge();
137     final SmName<S> varName = nameBridge.name(new QName("to"));
138     final GxVariantBridge<I, N, A, X> valueBridge = pcx.getVariantBridge();
139     final X value = valueBridge.stringValue("David");
140
141     transformer.bindVariableValue(varName, value);
142     transformer.bindVariableValue(nameBridge.name(new QName("http://www.example.com", "from")), valueBridge.stringValue("Julie"));
143
144     final N documentNode = transformer.transform(null, pcx);
145
146     final GxModel<N, A, S, T> model = pcx.getModel();
147
148     assertEquals(NodeKind.DOCUMENT, model.getNodeKind(documentNode));
149     final N email = model.getFirstChildElement(documentNode);
150     final N to = model.getFirstChildElementByName(email, nameBridge.symbolize("http://www.example.com"), nameBridge.symbolize("to"));
151     assertEquals("David", model.getStringValue(to));
152     final N from = model.getFirstChildElementByName(email, null, nameBridge.symbolize("from"));
153     assertEquals("Julie", model.getStringValue(from));
154     final N again = model.getFirstChildElementByName(email, nameBridge.symbolize("http://www.example.com"), null);
155     assertEquals("David", model.getStringValue(again));
156 }
157
158 public void skipExternalFunctions() throws ParserConfigurationException, IOException, GxException, ExprException, URISyntaxException {
159         {
160                 final GxProcessingContext<I, U, N, A, S, T, X> pcx = newProcessingContext();
161
162                 final Resolver resolver = getResolver();
163
164                 final Resolved<InputStream> xmlInput = resolver.resolveInputStream(new URI("exslt.xml"));
165
166                 final GxDocumentBuilderFactory<N, S> f = new DocumentBuilderFactory(I, U, N, A, S, T, X>(pcx);
167
168                 final GxDocumentBuilder<N> builder = f.newDocumentBuilder();
final N document = builder.parse(xmlInput.getResource(),
xmlInput.getSystemId());

final Resolved<InputStream> xslInput =
resolver.resolveInputStream(new URI("exslt.xsl"));

final GxTransformBuilder<I, U, N, A, S, T, X> compiler = new
XSLTransformBuilder<I, U, N, A, S, T, X>(pcx);

final String namespaceURI = "http://exslt.org/strings";
final ExsltStringsFunctionGroup<I, U, N, A, S, T, X>
functions = new ExsltStringsFunctionGroup<I, U, N, A, S, T,
X>(namespaceURI, pcx);

compiler.setFunctionSigns(namespaceURI, functions);
compiler.setFunctionImpls(namespaceURI, functions);

final GxTransform<I, U, N, A, S, T, X> compiled =
compiler.prepareTransform(xslInput.getResource(),
xslInput.getSystemId());

final GxSerializerFactory<I, U, N, A, S, T, X> sf = new
SerializerFactory<I, U, N, A, S, T, X>(pcx);

// TODO: Extract configuration.
// compiled.configure(sf);

sf.setIndent(true);

final StringWriter w = new StringWriter();

final GxSequenceHandler<A, S, T> handler =
sf.newSerializer(w);

final GxTransformer<I, U, N, A, S, T, X> transformer =
compiled.newTransformer();
transformer.transform(document, pcx, handler);

// System.out.println(w.toString());

public void skipHotel() throws ParserConfigurationException,
IOException, GxException, ExprException, URISyntaxException
{
final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();

final Resolver resolver = getResolver();

final Resolved<InputStream> xmlInput =
resolver.resolveInputStream(new URI("hotel.xml"));
final GxDocumentBuilderFactory<N, S> f = new DocumentBuilderFactory<I, U, A, S, T, X>(pcx);

final GxDocumentBuilder<N> builder = f.newDocumentBuilder();

final N document = builder.parse(xmlInput.getResource(), xmlInput.getSystemId());

final Resolved<InputStream> xslInput = resolver.resolveInputStream(new URI("hotel.xsl"));

final GxTransformBuilder<I, U, A, S, T, X> compiler = new XSLTransformBuilder<I, U, A, S, T, X>(pcx);

final GxTransform<I, U, A, S, T, X> compiled = compiler.prepareTransform(xslInput.getResource(), xslInput.getSystemId());

final GxTransformer<I, U, A, S, T, X> transformer = compiled.newTransformer();

final GxNameBridge<S> nameBridge = pcx.getNameBridge();
final SmName<S> varName = nameBridge.name(new QName("MessageData"));

final GxVariantBridge<I, A, S, X> valueBridge = pcx.getVariantBridge();
final X value = valueBridge.node(document);

transformer.bindVariableValue(varName, value);

final N documentNode = transformer.transform(null, pcx);

final GxModel<N, A, S, T> model = pcx.getModel();

equivalent(NodeKind.DOCUMENT, model.getNodeKind(documentNode));

final N searchHotelRequest = model.getFirstChildElement(documentNode);
final N parameters = model.getFirstChildElementByName(searchHotelRequest, nameBridge.symbolize("http://xmlns.example.com/1189038295781"), nameBridge.symbolize("parameters"));

final N searchHotel = model.getFirstChildElementByName(parameters, nameBridge.symbolize("http://www.xyzcorp/procureservice/QueryGDS_Europe/"), nameBridge.symbolize("searchHotel"));

final N country = model.getFirstChildElementByName(searchHotel, nameBridge.symbolize("http://www.xyzcorp/procureservice/QueryGDS_Europe/"), nameBridge.symbolize("country"));

assertEquals("USA", model.getStringValue(country));

XQuery

001 package org.gxml.book.xquery;
002
003 import java.io.StringWriter;
004 import java.math.BigInteger;
005 import java.net.URI;
006
007 import javax.xml.namespace.QName;
008
009 import org.gxml.book.common.SampleApp;
010 import org.gxml.sa.GxAtomBridge;
011 import org.gxml.sa.GxNameBridge;
012 import org.gxml.sa.GxProcessingContext;
013 import org.gxml.sa.GxSequenceHandler;
014 import org.gxml.sa.GxVariantBridge;
015 import org.gxml.xs.SmName;
016
017 import com.tibco.gxml.sa.api.common.lang.GxXQConnection;
018 import com.tibco.gxml.sa.api.common.lang.GxXQDataSource;
019 import com.tibco.gxml.sa.api.common.lang.GxXQExpression;
020 import com.tibco.gxml.sa.api.common.lang.GxXQPreparedExpression;
021 import com.tibco.gxml.sa.processor.serialization.api.GxSerializerFactory;
022 import com.tibco.gxml.sa.processor.serialization.impl.SerializerFactory;
023 import com.tibco.gxml.sa.processor.xquery.XQEngine;
024 import com.tibco.gxml.sa.processor.xquery.XQErrorCatcher;
025
026 /**
027  * Introduction to XQuery.
028  */
029 public abstract class XQuerySample<I, U, N extends I, A extends I,
S, T, X> extends SampleApp<I, U, N, A, S, T, X>
030 {
031     public void testExample() throws Exception
032     {
033         // Obtain a new processing context from the application.
034         final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();
035
036         final GxXQDataSource<I, U, N, A, S, T, X> ds = new
XQEngine<I, U, N, A, S, T, X>(pcx);
037
038         final GxXQConnection<I, U, N, A, S, T, X> conn =
ds.getConnection();
039
040         final String expression = "<x>{text{for $i in (1,2,3,4)
return $i * 2}}</x>";
final GxXQPreparedExpression<I, U, N, A, S, T, X> expr =
    conn.prepareExpression(expression);

final GxSerializerFactory<I, U, N, A, S, T, X> sf = new
    SerializerFactory<I, U, N, A, S, T, X>(pcx);
    sf.setMethod(new QName("xml"));
    sf.setOmitXmlDeclaration(true);
    final StringWriter sw = new StringWriter();
    final GxSequenceHandler<A, S, T> handler =
    sf.newSerializer(sw);

expr.executeQuery(handler);

final String actual = sw.toString();
    assertEquals(expression, "<x>2 4 6 8</x>", actual);

public void testGettingStarted() throws Exception
{
    // Obtain a new processing context from the application.
    final GxProcessingContext<I, U, N, A, S, T, X> pcx =
        newProcessingContext();

    final GxXQDataSource<I, U, N, A, S, T, X> ds = new
        XQEngine<I, U, N, A, S, T, X>(pcx);

    final GxXQConnection<I, U, N, A, S, T, X> conn =
        ds.getConnection();

    final GxXQExpression<I, U, N, A, S, T, X> expr =
        conn.createExpression();

    final String es = "for $n in fn:doc('catalog.xml')//item
        return fn:data($n/name)";

    final URI systemId = new URI("catalog.xml");

    expr.setBaseURI(systemId);

    @SuppressWarnings("unused")
    final X value = expr.executeQuery(es);

    public void testHelloWorld() throws Exception
{
    final GxProcessingContext<I, U, N, A, S, T, X> pcx =
        this.newProcessingContext();

    final GxXQDataSource<I, U, N, A, S, T, X> ds = new
        XQEngine<I, U, N, A, S, T, X>(pcx);
final GxXQConnection<I, U, N, A, S, T, X> conn =
ds.getConnection();

conn.setScriptingMode(true);

final String expression = "declare variable $x external;
concat('Hello, ', $x, '!'"");

final GxXQPreparedExpression<I, U, N, A, S, T, X> expr =
conn.prepareExpression(expression);

final GxSerializerFactory<I, U, N, A, S, T, X> sf = new
SerializerFactory<I, U, N, A, S, T, X>(pcx);

sf.setOmitXmlDeclaration(true);
sf.setIndent(false);
sf.setMethod(new QName("xml"));

final StringWriter sw = new StringWriter();

final GxSequenceHandler<A, S, T> handler =
sf.newSerializer(sw);

final GxNameBridge<S> nameBridge = pcx.getNameBridge();
final GxVariantBridge<I, N, A, X> valueBridge =
pcx.getVariantBridge();

final SmName<S> varName = new
SmName<S>(nameBridge.symbolize("x"), nameBridge);
final X value = valueBridge.stringValue("World");

expr.bindVariableValue(varName, value);

expr.executeQuery(handler);

String actual = sw.toString();
assertEquals(expression, "Hello, World!", actual);

// Obtain a new processing context from the application.
final GxProcessingContext<I, U, N, A, S, T, X> pcx =
newProcessingContext();

final GxXQDataSource<I, U, N, A, S, T, X> ds = new
XQEngine<I, U, N, A, S, T, X>(pcx);

final GxXQConnection<I, U, N, A, S, T, X> conn =
ds.getConnection();

// final String expression = "";
final String expression = "count((element elem {1, 'string',
1,2e3})/text())";
final GxXQPreparedExpression<I, U, N, A, S, T, X> expr = conn.prepareExpression(expression);

final GxSerializerFactory<I, U, N, A, S, T, X> sf = new SerializerFactory<I, U, N, A, S, T, X>(pcx);
sf.setMethod(new QName("xml"));
sf.setOmitXmlDeclaration(true);
final StringWriter sw = new StringWriter();
final GxSequenceHandler<A, S, T> handler = sf.newSerializer(sw);
expr.executeQuery(handler);

final String actual = sw.toString();
assertEquals(expression, "1", actual);
}

public void testProblem() throws Exception
{
    final GxProcessingContext<I, U, N, A, S, T, X> pcx = this.newProcessingContext();

    final GxXQDataSource<I, U, N, A, S, T, X> ds = new XQEngine<I, U, N, A, S, T, X>(pcx);

    final GxXQConnection<I, U, N, A, S, T, X> conn = ds.getConnection();

    final XQErrorCatcher messages = new XQErrorCatcher();
    conn.setErrorHandler(messages);
    conn.setCompatibleMode(false);
    conn.setScriptingMode(true);

    final String expression = "(xs:untypedAtomic('1'),xs:untypedAtomic('2')) = (xs:untypedAtomic('2.0'),2.0)";

    final GxXQPreparedExpression<I, U, N, A, S, T, X> expr = conn.prepareExpression(expression);
    final X value = expr.executeQuery();
    final GxVariantBridge<I, N, A, X> variantBridge = pcx.getVariantBridge();
    switch (variantBridge.getNature(value))
    {
        case ITEMS:
        {
            @SuppressWarnings("unused")
            final Iterable<I> items = variantBridge.getItemSet(value);
165                      // System.out.println(items);
166                       }
167                     break;
168                    case ATOM:
169                        {
170                            @SuppressWarnings("unused")
171                            final A atom = variantBridge.getAtom(value);
172                            @SuppressWarnings("unused")
173                            final GxAtomBridge<A, S> atomBridge =
174                                pcx.getAtomBridge();
175                            // System.out.println(atomBridge.getC14NForm(atom));
176                        }
177                     break;
178                    case STRING:
179                        {
180                            @SuppressWarnings("unused")
181                            final String strval = variantBridge.getString(value);
182                            // System.out.println(strval);
183                        }
184                     break;
185                    case INTEGER:
186                        {
187                            @SuppressWarnings("unused")
188                            final BigInteger integer =
189                                variantBridge.getInteger(value);
190                            // System.out.println(integer);
191                        }
192                     break;
193                    default:
194                        {
195                            throw new
196                                AssertionError(variantBridge.getNature(value));
197                        }
198                   }
199               public void testTyping() throws Exception
200               {
201                     final GxProcessingContext<I, U, N, A, S, T, X> pcx =
202                     this.newProcessingContext();
203               final GxXQDataSource<I, U, N, A, S, T, X> ds = new
204                     XQEngine<I, U, N, A, S, T, X>(pcx);
205               final GxXQConnection<I, U, N, A, S, T, X> conn =
206                     ds.getConnection();
207                     conn.setScriptingMode(true);
208               final XQErrorCatcher messages = new XQErrorCatcher();
209
210         conn.setErrorHandler(messages);
211         final String expression = "declare variable $x external;
212         contains(string(number($x)),'NaN')";
213         final GxXQPreparedExpression<I, U, N, A, S, T, X> expr =
214         conn.prepareExpression(expression);
215         final GxSerializerFactory<I, U, N, A, S, T, X> sf = new
216         SerializerFactory<I, U, N, A, S, T, X>(pcx);
217         sf.setOmitXmlDeclaration(true);
218         sf.setIndent(false);
219         sf.setMethod(new QName("xml"));
220         final StringWriter sw = new StringWriter();
221         final GxSequenceHandler<A, S, T> handler =
222         sf.newSerializer(sw);
223         final GxNameBridge<S> nameBridge = pcx.getNameBridge();
224         final GxVariantBridge<I, N, A, X> valueBridge =
225         pcx.getVariantBridge();
226         final SmName<S> varName = new
227         SmName<S>(nameBridge.symbolize("x"), nameBridge);
228         final X value = valueBridge.doubleValue(5.0);
229         expr.bindVariableValue(varName, value);
230         expr.executeQuery(handler);
231         String actual = sw.toString();
232         assertEquals(expression, "false", actual);
233     }
234 }

Validation

001 package org.gxml.book.validation;
002
003 import java.io.InputStream;
004 import java.net.URI;
005 import java.util.LinkedList;
006 import java.util.List;
007
008 import org.gxml.book.common.SampleApp;
009 import org.gxml.sa.GxFragmentBuilder;
010 import org.gxml.sa.GxModel;
011 import org.gxml.sa.GxProcessingContext;
012 import org.gxml.xdm.Resolved;
013 import org.gxml.xdm.Resolver;
{
    public void testByteStreamValidation() throws Exception
    {
        // Load a top-level schema into the processing context.
        final List<Resolved<InputStream>> resources = new LinkedList<Resolved<InputStream>>();
        resources.add(getResolver().resolveInputStream(new URI("PurchaseOrder.xsd")));

        final SmExceptionCatcher errors = new SmExceptionCatcher();
        final SmMetaLoadArgs args = new SmMetaLoadArgs();

        final GxProcessingContext<I, U, N, A, S, T, X> pcx = newProcessingContext();

        final W3cXmlSchemaParser<A, S> parser = new W3cXmlSchemaParser<A, S>(new SmAtomBridgeOnGxAtomBridgeAdapter<A, S>(pcx.getAtomBridge()));

        for (final Resolved<InputStream> resource : resources)
        {
            pcx.register(parser.parse(resource.getLocation(), resource.getResource(), resource.getSystemId(), errors, args, pcx));
        }
        pcx.lock();

        // Create a validator...
        final GxValidatorCacheFactory<A, S, T> vcf = new ValidatorCacheFactory<I, U, N, A, S, T, X>(pcx);
        final GxValidatorCache<A, S, T> vc = vcf.newValidatorCache();
        }
final GxContentValidator<A, S, T> validator =
vc.newContentValidator();

// Set the downstream event handler which contains annotations and typed content.
validator.setGxContentHandler(/* ...*/null);
validator.setExceptionHandler(errors);

// The document node that we wish to validate.
final Resolved<InputStream> xmlInput =
getResolver().resolveInputStream(new URI("PurchaseOrder.xml"));

final GxDocumentBuilderFactory<N, S> factory = new
DocumentBuilderFactory<I, U, N, A, S, T, X>(pcx);

final GxDocumentBuilder<N> builder =
factory.newDocumentBuilder();

final N document = builder.parse(xmlInput.getResource(),
xmlInput.getSystemId());

// Stream the document into the validator.
final GxModel<N, A, S, T> model = pcx.getModel();
model.stream(document, true, true, validator);

if (errors.size() > 0)
{
    // You've got errors.'
}

public void testTreeValidation() throws Exception
{
    final Resolver resolver = getResolver();

    // Load a top-level schema into the processing context.
    final List<Resolved<InputStream>> resources = new
    LinkedList<Resolved<InputStream>>();
    resources.add(getResolver().resolveInputStream(new
    URI("PurchaseOrder.xsd")));

    final GxProcessingContext<I, U, N, A, S, T, X> pcx =
    new ProcessingContext();
    final W3cXmlSchemaParser<A, S> parser = new
    W3cXmlSchemaParser<A, S>(new SmAtomBridgeOnGxAtomBridgeAdapter<A,
    S>(pcx.getAtomBridge()));
    for (final Resolved<InputStream> resource : resources)
pcx.register(parser.parse(resource.getLocation(),
resource.getResource(), resource.getSystemId(), errors, args, pcx));
}

pcx.lock();
// The document node that we wish to validate.
@SuppressWarnings("unused")
final URI xmlLocation = new URI("PurchaseOrder.xml");
final URI xmlSystemId = new URI("PurchaseOrder.xml");
final Resolved<InputStream> xmlInput = resolver.resolveInputStream(xmlSystemId);

final GxDocumentBuilderFactory<N, S> factory = new
DocumentBuilderFactory<I, U, N, A, S, T, X>(pcx);

final GxDocumentBuilder<N> builder =
factory.newDocumentBuilder();

final N documentIn = builder.parse(xmlInput.getResource(),
xmlInput.getSystemId());

@SuppressWarnings("unused")
final N documentOut = validate(documentIn, errors, pcx);

if (errors.size() > 0)
{
    // You've got errors.'
    for (@SuppressWarnings("unused")
    final SmException error : errors)
    {
        // System.out.println(error.getLocalizedMessage());
    }
}

/**
 * This static function illustrates a helper function for
validating a document tree. <br/>
 * Note that we assume that the processing context is already
loaded with meta-data.
 * @param node
 * The input document.
 * @param errors
 * The error handler.
 * @param pcx
 * The processing context.
 */
public static <I, U, N extends I, A extends I, S, T, X> N
validate(final N node, final SmExceptionHandler errors, final
GxProcessingContext<I, U, N, A, S, T, X> pcx)
final GxValidatorCacheFactory<A, S, T> vcf = new ValidatorCacheFactory<I, U, N, A, S, T, X>(pcx);

// We already have a tree as input so we'll use the content validator'
// and stream the document in as a bunch of events (a bit like SAX, but not lexical).
final GxValidatorCache<A, S, T> vc = vcf.newValidatorCache();

final GxContentValidator<A, S, T> validator = vc.newContentValidator();

validator.setExceptionHandler(errors);

final GxModel<N, A, S, T> model = pcx.getModel();

// We want to produce a node so we'll need a fragment builder at the output.'
final GxFragmentBuilder<N, A, S, T> builder = pcx.newFragmentBuilder();

// Connect the pieces together so that the validation output builds a tree.
validator.setGxContentHandler(builder);

// Make it so!
model.stream(node, true, true, validator);

// Practice safe coding: We don't know what might happen if there are errors.'
final List<? extends N> nodes = builder.getNodes();
if (nodes.size() > 0)
{
    return nodes.get(0);
}
else
{
    return null;
}
}
Index

A

adding
  comments in XPath and XSLT 99, 104
data to the mediation exchange 11
interfaces to mediation flows 23, 23
mediation tasks to a path 40
routing cases 62
the same interface multiple times to a mediation flow 6
addressing schema elements 102
automatically mediating target and mediation interfaces 25
AutoMediate ANT command
  examples 224
  exceptions 225
  syntax and options 224
AutoMediate command
  creating a SOA project 217
  examples 222
  exceptions 222
  running 218
  syntax and options 220

B

binding mediation and target interfaces to actual services 42
boolean values, testing in XPath 104
business logic 4
  mediation tasks 8

C

Catch Fault task 139, 168
  faults from target operation 143
Choose statement in XSLT 95
command, running the AutoMediate 218
comments 99
comments in XPath 104
components, mediation 42
Composite Editor 42
composites 42
constants in XPath 87
contributing data to the mediation exchange 11, 52, 78, 80
copying and pasting mediation tasks 41
creating local WSDL files 25, 25, 25
creating mediation flows 16, 16
  from existing web services 17
custom mediation tasks
  migrating to version 2.1 or 2.2 148
custom xpath functions
  creating 89
  deploying 92
customer support xxii, xxii

D

data
  in mediation flows 10
  transforming message data 72
datatypes
  in schema components 82
  mediation flow properties 21
  of route task variables 65
Index

deleting
  interfaces 25
  mappings 93
  mediation tasks 41, 41
  paths 27
  XSLT statements 93
deploying projects 13
designing mediation flows 12

designing mediation flows 12

evaluating mediation flows 12

developing mediation flows 12

deploying projects 13

designing mediation flows 12

deploying projects 13

designing mediation flows 12

deploying projects 13

designing mediation flows 12

deploying projects 13

E
End Mediation task 170
evaluation context 103
exceptions 222, 225
exchange variable
  defining 35
  logging 52
  setting 36

F
fault path 6, 138, 143
  effect of a route task 59
For Each statement in XSLT 97
functions in XPath 87

G
Generate Reply task 172
generated artifacts
  mediation flow 216

H
Handle Reply task 173

handling faults 138
  from the mediation flow 144
  from the target operation 143
  returning faults to the invoker 146
-host option 220
-httpConnectorname option 221

I
icon size 20
If statement in XSLT 96
in-only message exchange pattern 5
in-out message exchange pattern 5
input path 6, 27
  effect of a route task 59
interfaces 5, 23, 25
  adding to mediation flows 23
  deleting 25
Invoke Operation task 174

L
location of web services in project 23
Log task 176
  configuring 53
  overview of 52
M

mapping 100
  addressing schema elements 102
comments 99
deleting 93
example 75
hints in the message panel 100
repairing 100
variable data in route tasks 65
XSLT Choose statement 95
XSLT For Each statement 97
XSLT If statement 96
XSLT variables 98
mediation
  components 2, 42
  exchange 10
  flows, about 4
  overview 2
mediation flow
  catching faults from 144
Mediation Flow Editor 4, 8, 20
  paths 7
  preferences 20
mediation flow, generated 216
mediation flows 16
  adding interfaces 23
  creating 16, 16
  creating from existing web services 17, 17
  designing 12, 12
  handling faults 138
interfaces 5
logging properties 52
multiple routes 8, 64, 189
paths 6
properties 21
relationship to components 42
route tasks 7
throwing faults 141
transforming message data 72
Mediation Flows special folder 16, 16, 16
mediation interface 5, 25
  adding 25
  adding the same interface multiple times 6
  automatically mediating 25
  binding to services 42
  component service 42
  deleting 25
mediation operations
  input path requirement 27
mediation path 6
  effects of a route task 59
mediation tasks 8, 8, 8
  adding to a path 40
  Catch Fault 168
  Catch Fault task 139
  changing icon size 20
  contributing data to the mediation exchange 11
  copying and pasting 41, 41
  creating and deploying 147
  deleting from a path 41, 41
  End Mediation 9, 170
  Generate Reply 9, 172
  Handle Reply 9, 173
  Invoke Operation 9, 174
  Log 8, 176
  Log task 52
  Parse XML 8, 180
  Query Database 8, 183
  Render XML 8, 187
  Route 9, 189
  Route task 58
  Send Fault 193
  Send Fault task 146
  Set Context 9, 194
  Set Dynamic Reference 9
  Set Exchange Variable 9, 200
  Throw Fault 9, 202
  Throw Fault task 141
  Transform 8, 72, 203
  Validate XML 9, 206
  XPath Route 9, 211
  XPath Route task 58
message context 11
   logging 52
   properties 28
   schema 84
message data
   logging 52
message exchange patterns 5, 5, 7
messages 10
   routing to a set of target operations 59
   transforming data 72
multiple routes in Route task 189

N

namespaces in XPath expressions 103

O

offering different quality to each client 24
one-way message exchange pattern 5
optional schema elements 83
options
   -host 220
   -httpConnectorName 221
   -port 220
   -projectDir 220
   -projectName 220
   -refJmsConnectionFactoryJndiName 221
   -refJmsDestinationJndiName 221
   -serviceJmsConnectionFactoryJndiName 221
   -serviceJmsDestinationJndiName 221
output path 6, 27
   effect of a route task 59
overriding mediation flow properties 21
overview 2

P

Parse XML task 180

pasting mediation tasks 41
paths 6, 41
   adding tasks 40
   copying tasks 41
   deleting 27
   deleting tasks 41, 41
   effect of deleting interfaces 25
   fault path 138
   mediation flows 6
   mediation tasks 8
   message exchange patterns 6
   sub-paths 59
   working with 27, 27
planning mediation flows 6
   -port option 220
   preferences 20, 20
   project
      created by AutoMediate tool 216
      how AutoMediate creates 217
   -projectDir option 220
   projection icon 25
   -projectName option 220
   projects 4
      Mediation Flows special folder 16
      Service Descriptors special folder 23
properties 21, 21

Q

quality of service 24
Query Database task 183

R

refJmsConnectionFactoryJndiName option 221
refJmsDestinationJndiName option 221
relationship between components and items in a mediation flow 42
removing mappings and statements in Transform tasks 93
Render XML task 187
Index | 287

repairing mappings 100
repeating schema elements 83
request-response message exchange pattern 5
Route task 58, 189
  adding or deleting variables 65
  conditions 67
  effect on paths 59
  mapping data to variables 65
  modifying target destinations 63
  multiple routes 8, 64, 189, 189
  routing cases 62
  variables 65
running projects 13

S

schemas 82
  addressing components 102
  qualifier characters 83
  transforming data 72
search predicates in XPath expressions 103
Send Fault task 146, 193
service assemblies 12
Service Descriptors special folder 23, 23, 23
serviceJmsConFactoryJndiName option 221
serviceJmsDestJndiName option 221
service-oriented architecture (SOA) 2
Set Context task 194
Set Exchange Variable task 200
setting properties on mediation flows 21
Show Edit Tab XPath window 93
size of mediation task icons 20
SOA Projects 216
special folders 16, 23
sub-paths 59
support, contacting xxii, xxii
syntax of AutoMediate ANT command 224
syntax of AutoMediate command 220

T

target interface 5
  adding 25
  adding the same interface multiple times 6
  automatically mediating 25
  binding to references 42
  component references 42
  deleting 25
target operation
  catching faults 143
  handling faults 138
tasks 8, 8
  adding to a path 40, 40
  Catch Fault 139, 168
  contributing data to the mediation exchange 11
  copying and pasting 41
  deleting from a path 41
  End Mediation 9, 170
  Generate Reply 9, 172
  Handle Reply 9, 173
  Invoke Operation 8, 174
  Log 8, 52, 176
  Parse XML 8, 180
  Query Database 8, 183
  Render XML 8, 187
  Route 9, 58, 189
  Send Fault 193
  SendFault 146
  Set Context 194
  Set context 9
  Set Dynamic Reference 9
  Set Exchange Variable 9, 200
  Throw Fault 9, 141, 202
  Transform 8, 72, 203
  Validate XML 9, 206
  XPath Route 9, 211
  XPathRoute 58
technical support xxii, xxii
Throw Fault task 141, 202
TIBCO ActiveMatrix SOA projects 4
deploying and running 13, 13
  Mediation Flows special folder 16
  Service Descriptors special folder 23
TIBCO_HOME xix
Transform task 72, 203
   comments 99
Context panel 84
   example 74
icons in toolbar 93
message panel 86
right-click menu 94
XSLT Choose statement 95
XSLT For Each statement 97
XSLT If statement 96
XSLT variables 98

WSDL
   @name attribute 220
   and -projectName option 221
   as input 216, 217
   concrete 218
   location 220, 221
   source location 221
   specifying 218
   URI format 220, 224
WSDL files 5, 25

U

URI format for WSDL 220, 224

V

Validate XML task 206
   variables
      in route tasks 65
      in XSLT 98
viewing paths 7
virtualization 2

W

web services 5, 5
   binding to mediation and target interfaces 42
   creating mediation flows from 17
   interfaces in mediation flows 23
   working with paths 27

X

XML
   converting to string or binary 105, 187
   parsing to a tree 105, 180
   transforming an external XSLT 79
   using XPath to route messages 211
   validating a schema 105, 206
XPath 72, 102
   evaluation context 103
   functions and constants 87
   search predicates 103
   testing boolean values 104
XPath Route task 58, 211
   adding or deleting variables 65
   conditions 68
   effect on path 59
   mapping data to variables 65
   modifying target destinations 63
   routing cases 62
   variables 65
XSLT 72
   Choose statement 95
   comments 99
   deleting statements 93
   For Each statement 97
   hints 100
   If statement 96
   variables 98